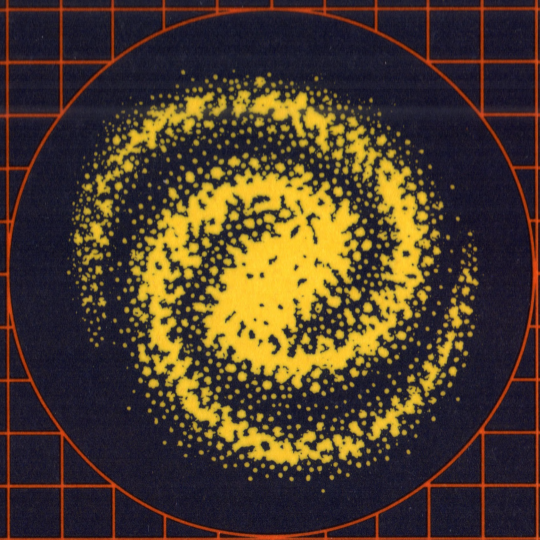
A+L AG

Modula-2
Software-
Entwicklungssystem

M2 AMIGA™



Benutzer Handbuch

Software und Dokumentation: René Degen, Claudio Nieder, Markus Schaub, Jörg Straube

Vertrieb: A+L AG, Schweiz

© Copyright 1988

A+L AG
Im Späten 23
CH-8906 Bonstetten

Die vorliegende Dokumentation ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, besonders die der Übersetzung, des Nachdrucks, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege, der Speicherung und Auswertung in Datenverarbeitungsanlagen, bleiben, auch bei Verwertung von Teilen der Dokumentation, dem Herausgeber vorbehalten.

Zur Beachtung: A+L AG lehnt jede Art von Garantien ab, welche die Tauglichkeit der Software für einen bestimmten Zweck versprechen. A+L AG haftet nicht für Fehler in dieser Dokumentation und auch nicht für Schäden, die durch den Gebrauch der Dokumentation, deren Inhalt oder von A+L-Software direkt oder indirekt entstehen.

A+L AG behält sich vor, jederzeit Änderungen an dieser Dokumentation oder deren Software vorzunehmen, ohne jeden Anwender davon zu benachrichtigen.

Herstellung und Vertrieb:

A+L AG
Im Späten 23
CH-8906 Bonstetten

Vertrieb:

RTA - Real Time Associates Ltd.
Canning House
59, Canning Road
GB-Croydon, Surrey, CRO 6QF

Interface Technologies Corporation
3336 Richmond, Suite 323
USA-Houston, TX 77098-9990

ISBN 3-907017-20-X

Printed in Switzerland

Kundenunterstützung

Lesen Sie diese Information, bevor Sie das Siegel des Diskettenumschlags brechen.

Das Siegel des Diskettenumschlags brechen bedeutet Akzeptieren der Bedingungen des A+L-Lizenzvertrags.

A+L AG und deren Partner helfen Ihnen, Ihr Programm optimal einzusetzen. Wir empfehlen Ihnen, die folgenden Informationen über Registration, Unterstützung und Ihre Rechte und Pflichten als Kunde durchzulesen.

A+L AG und deren Partner (dort wo Sie das Produkt gekauft haben) bieten Ihnen telefonische Unterstützung durch einen Kundendienst. Egal wie kompliziert Ihre Frage oder Ihr Problem ist, wir versuchen Ihnen rasch zu helfen. Bevor Sie jedoch anrufen, vergewissern Sie sich, dass Sie Ihre Registrationskarte/Lizenzvertrag eingeschickt haben. Andernfalls ist die A+L AG davon entbunden, Ihnen Auskunft zu geben.

Falls Probleme mit Ihrem Programm auftauchen, gehen Sie wie folgt vor:

1. Versuchen Sie das Problem mit diesem Handbuch zu lösen.

2. Rufen Sie A+L oder Ihren Händler zwischen 9 und 17 Uhr an. Sie sollten dann Ihre Programm-Serienummer zur Hand haben.

Beschränkte Garantie

A+L AG garantiert, dass:

1. das Material, Disketten und Dokumentation nicht beschädigt sind.

2. das Programm ordnungsgemäss auf die Diskette(n) kopiert ist.

3. die Dokumentation vollständig ist und alle Informationen enthält, die A+L AG für die Bedienung des Programms als erforderlich hält.

4. das Programm im Prinzip so funktioniert, wie es im Handbuch beschrieben ist.

Falls Sie uns innert von 30 Tagen nach Auslieferung Fehler schriftlich melden oder uns defektes Material zustellen, steht Ihnen im Rahmen dieser beschränkten Garantie das Recht zu, Disketten oder Software ersetzt zu bekommen. Legen Sie der schriftlichen Fehlermeldung das entsprechende Rückporto in Ihrer Währung bei. Das Recht auf Ersatz besteht nur, wenn wir im Besitze Ihres unterschriebenen Lizenzvertrages sind. Senden Sie diesen sowie allfällige Fehlermeldungen oder defekte Disketten an Ihren Händler oder an folgende Adresse:

A+L AG
Im Späten 23
CH-8906 Bonstetten
Schweiz

A+L AG haftet nicht für direkte oder indirekte Schäden.

A+L AG lehnt jede Art von Haftung oder stillschweigender Garantie ab, insbesondere, dass sich die Produkte von A+L AG für einen ganz bestimmten Zweck eignen. A+L AG beschränkt ihre Garantie auf das Ersetzen defekter Disketten und Programme.

Entsprechend obenstehender Angaben tragen Sie als Anwender die Verantwortung, mit Ihrem Programm sorgfältig umzugehen, und damit auch die Kosten mutwilliger Beschädigungen und Fehler ausserhalb der obgenannten Beschränkungen.

Anwenderlizenz

Erinnern Sie sich daran, dass das Aufbrechen des Siegels auf dem Diskettenumschlag das Akzeptieren der Bedingungen dieser Lizenz bedeutet.

A+L AG behältt sich alle Rechte für ihre Programme vor. Jedes Programm fällt unter diesen Lizenzvertrag. Ein entsprechendes Vergehen wird geahndet.

1. **Erlaubter Gebrauch:** Das Programm darf nur auf einem einzigen Computer mit einem einzigen Bildschirm verwendet werden. Sie dürfen eine oder mehrere Kopien dieses Programms herstellen, damit Sie über ausreichend Kopiedisketten verfügen, falls Ihre Arbeitsdisketten zerstört werden. Dazu dürfen Sie eine und nur eine Kopie des Programms auf Ihrer Harddisk machen.

2. **Unerlaubter Gebrauch:** Ohne ausdrückliche schriftliche Bewilligung von A+L AG dürfen Sie folgendes *nicht* tun:

   - Das Programm in Verbindung mit mehr als einem Computer gleichzeitig verwenden.
   - Das Programm, Kopien davon oder dessen Dokumentation an jemand anderes verkaufen.
   - Erstellen von Kopien des Programms, der Dokumentation oder von Disketten, ausgenommen derjenigen Kopien, die in diesem Lizenzvertrag bewilligt sind.
   - Die Verwendung des Programms in einem Netzwerk, einem Time-Sharing-System, einem interaktiven Fernsehnetzwerk, einem Mehrprozessorsystem oder einem Mehrplatzsystem falls nicht eine spezielle Lizenz von A+L vorliegt, beziehungsweise jeder einzelne Benutzer über eine eigene Lizenz mit A+L verfügt.
   - Veränderungen am Programm vornehmen.
   - Das Übertragen des Programms oder das Bewilligen einer Unterlizenz, Vermietung oder das Übertragen weiterer Rechte auf andere.
   - Eine wörtliche oder übertragene Übersetzung der Dokumentation oder des Programms herstellen.
   - Anpassen des Programms an eine andere Hardware.
   - Durchführen telefonischer oder elektronischer Datenübertragung des Programms.
   - Vertrieb oder Vermietung des Programms an andere auf dauernder oder auch nur vorübergehender Basis.

Beendigung der Lizenz. Die Lizenz gilt als beendet, wenn Sie alle Disketten, Dokumente und Kopien aller Art zerstören. Diese Lizenz fällt ebenfalls dahin, wenn Sie gegen eine der obgenannten Bedingungen verstossen. Sie sind in einem solchen Fall verpflichtet, alle Ihre Disketten, Dokumente und Kopien aller Art zu vernichten.

Amiga	is a registered trademark of Commodore-Amiga, Inc.
AmigaDOS	is a trademark of Commodore-Amiga, Inc.
Kickstart	is a trademark of Commodore-Amiga, Inc.
Intuition	is a trademark of Commodore-Amiga, Inc.
Workbench	is a trademark of Commodore-Amiga, Inc.

Inhaltsverzeichnis

1 Amiga Modula-2

Einleitung 3

Was bietet dieses Handbuch? 4

Aufbau 4

Notation 6

Terminologie 6

Verweise 8

2 Installation

Inhalt der Disketten 3

Dateien geringerer Bedeutung 5

Dateien mit fester Anordnung 7

Andere Dateien 7

Der erste Schritt 9

Installation auf Floppy Disk 9

Installation auf Festplatte 12

Wie geht es weiter? 12

3 Arbeitsumgebung Amiga Modula-2

Übersicht 3

Programmaufruf 3

Optionen 4

Argumente 4

Hilfe 5

Das Projekt-Konzept 6

Einleitung 6
Details zum Projekt 7
Zusammenspiel verschiedener Projekte 9
m2project 10

4 Der Editor

Aufruf 3
Charakteristik des Editors 3
Spezielle Terminologie 3
Grundlegendes 5
Die Maus als Eingabegerät 6
Lesen und Abspeichern von Dateien 7
Löschen und Wiedereinsetzen von Textbereichen 9
Suchen und Ersetzen eines Textes 12
Ändern eines Textbereichs 13
Schalterstellungen 14
Bildschirminhalt auswählen und Fensterbehandlung 15
Makrobefehle 16
Modula-2 Befehle 17
Verschiedene Befehle 18
Betriebssystemumgebung 20
Optionen beim Aufruf 20
Rückgabewerte 21
Ein Beispiel 21
Befehlszusammenfassung 23

5 Der Compiler

Aufruf 3
Arbeitsweise des Compilers 3
Optionen 5
Einschränkungen 9
Rückgabewerte 14

6 Der Linker

Aufruf 3
Arbeitsweise des Linkers 3
Optionen 3
Fehlermeldungen 4
Rückgabewerte 5

7 Der Fehlerlister

Aufruf 3
Aufgabe und Arbeitsweise des Fehlerlisters 3
Optionen 3
Beispiel-Ausgaben 4
Fehlermeldungen 6
Rückgabewerte 6

8 Anmerkungen zu Amiga Modula-2

Eigenschaften 3
Ganze Zahlen 3
Gleitpunkt-Zahlen 5

Restriktionen des Einpass-Compilers 6
Anordnung (Alignment) 6
Sonstiges 8
Systemspezifische Erweiterungen / SYSTEM 9

9 Das Laufzeitsystem

Ein- und Ausgangscode 3
Compilerunterstützung 4
Die Behandlung von Laufzeitfehlern 5
Unterbrechungsmöglichkeit 6
Die Abschlussprozeduren 7
Die Debugprozeduren 9
Das Konzept der Aktivierungsstufen (Levelkonzept) 10
Fehlermeldungen des Laufzeitsystems 12
Prozessor Trap 12
^C (Benutzer Unterbruch) 13
Fehler beim Öffnen der xxx.library 13
Programmierter Halt 13
Ungültiger Case-Index 13
Funktion ohne RETURN beendet 14
Stapelüberlauf 14
Ungültiger Aufruf von Arts.Call 14
Adress-Fehler 15
Ungültige Instruktion 15
Division durch Null 15
Bereichsfehler 16
Überlauf 17

10 Technische Angaben

Von M2Amiga verwendete Dateiformate 3
Objektdatei 6
Symbol-/Referenzdatei 8
Beispiel 13
Fehlerdatei 17

Laufzeitumgebung von M2Amiga 18

Kontrollcode 21

11 Programmierhinweise

Übersetzung von C-Programmen 3
Aufzählungs- und Mengentypen 3
VAR-Parameter 4
Record-Varianten 6
Funktionsresultate 7

M2Amiga Besonderheiten 8
Zeichenketten 8
Terminate statt Exit 9
Verwendung von Abschlussprozeduren und Assert 9

12 Bibliotheken

Arguments 3

Arts 7

ASCII 11

Assembler 12

Conversions 17

Coroutines 20

FFPConversions 22

FFPInOut 24
FileMessage 25
FileNames 26
FileSystem 28
Erkennen des Dateiendes bei Leseoperationen 31
Heap 33
InOut 36
LongRealConversions 39
LongRealInOut 41
MathLib0 42
MathLibFFP 44
MathLibLong 46
RandomNumber 47
RealConversions 48
RealInOut 50
Scan 51
Storage 52
Str 53
Strings 56
SYSTEM 58
Terminal 59
Verwendeter Ein- und Ausgabekanal 61
Windows 64

13 Schnittstellen-Moduln

Audio 3

BootBlock 4

Clipboard 5

Console 6

ConUnit 8

DiskFont 10

Dos 12

Exec 19

ExecSupport 29

Expansion 30

GamePort 34

GfxMacros 35

Graphics 36

Hardware 50

Icon 55

Input 56

InputEvent 57

Intuition 59

Keyboard 77

KeyMap 78

Layers 80

MathIEEEDoubTrans 82

MathTrans 83

Narrator 84
Parallel 86
Printer 87
PrtBase 90
Resources 92
Serial 95
Timer 97
TrackDisk 98
Translator 100
Workbench 101

14 Cross-Reference

1 Amiga Modula-2

Einleitung 3

Was bietet dieses Handbuch? 4

Aufbau 4

Notation 6

Terminologie 6

Verweise 8

Einleitung

Modula-2 ist eine der neusten Sprachschöpfungen von Prof. Niklaus Wirth. Niklaus Wirth hat schon bei der Sprache Algol mitgewirkt, ferner eine Algol-Variation Algol–W vorgestellt, und er konnte seinen bisher grössten Erfolg mit der Sprache Pascal verbuchen. Dieser Erfolg scheint langsam von Modula-2 übertroffen zu werden, was nicht weiter verwunderlich ist, denn Modula-2 wird oft als Nachfolger von Pascal bezeichnet, aller Mängel bereinigt und um die Modularität und der systemnahen Möglichkeiten erweitert.

Doch Modula-2 ist weit mehr. Gleichzeitig mit der Sprachdefinition arbeitete Niklaus Wirth am Institut für Informatik an der ETH Zürich auch an einem Computer, der nur eine einzige Sprache versteht: Modula-2. Das Ziel war, alle Teile des Betriebssystems, sowie alle Anwendungsprogramme in Modula-2 zu programmieren. Im Jahre 1981 stellte Niklaus Wirth diesen "Modula-2 Computer" der Öffentlichkeit vor.

Seither ist oft genug bewiesen worden, dass Modula-2 auch für andere Rechner eine leistungsfähige Sprache ist. Die Klarheit der Sprache, die mögliche Aufteilung eines Programms und die trotzdem garantierte Verträglichkeitsprüfung machten Modula-2 zu einer oft verwendeten Entwicklungsumgebung auch für Grossprojekte.

Im Jahre 1985 stellte Niklaus Wirth am Institut für Informatik einen neuen Compiler vor: Kleine Änderungen der Sprachdefinition von Modula-2 machten es möglich, Modula-2 Programme in *einem* Durchgang zu compilieren. Das Resultat ist ein extrem schneller Compiler, der verhältnismässig klein ist. Dass das Einpass-Konzept keine entscheidende Einschränkung ist, zeigt sich in der Tatsache, dass der Compiler sich selbst compiliert.

Der Amiga Modula-2 Compiler basiert auf diesem Compiler. Er wurde speziell auf den Amiga zugeschnitten (Benutzerumgebung, Codeformat usw.) und in ein komplettes System eingebettet. Das Resultat ist ein Entwicklungssystem, das schnell und sauber arbeitet, sich sehr gut in die Amiga-Umgebung integriert und hoffentlich auch Ihnen Freude bereiten wird.

Was bietet dieses Handbuch?

Es beschreibt die Bedienung des Amiga Modula-2-Systems und richtet sich an diejenigen Programmierer, die bereits mit Modula-2 vertraut sind, oder die mit Hilfe eines Einführungsbuchs erste Schritte auf einem Computer wagen.

Es werden alle zum System gehörenden Komponenten ausführlich beschrieben. Dies sind ein Compiler, ein Linker, ein Editor, zwei Hilfsprogramme und alle Definitionen und Betriebssystemschnittstellen.

Dieses Handbuch ist weder ein Einführungs-, noch ein Nachschlagebuch für die Sprache Modula-2. Es ersetzt keinesfalls eine Dokumentation über den Commodore Amiga. Die Kommentare in den Schnittstellenbeschreibungen sind äusserst knapp gehalten, man konsultiere die entsprechenden Handbücher für diese Routinen.

Aufbau

Das Handbuch ist in 14 Teile gegliedert. Jeder Teil führt ein Inhaltsverzeichnis.

- **Amiga Modula-2:** Eine Einführung in das vorliegende Handbuch. Zeigt im wesentlichen dessen Benutzung.

- **Installation:** Eine Installationsanleitung für Amiga Modula-2. Bitte lesen Sie diesen Teil durch!

- **Arbeitsumgebung Amiga Modula-2:** Allgemeingültige Angaben über Benutzerführung der Programme. Deckt auch erste Unterschiede zwischen CLI und Workbench auf.

- **Editor:** Eine Anleitung zu m2emacs, dem mitgelieferten Editor. Ein äusserst mächtiger Editor, dessen Verwendung empfohlen wird. Auch ohne Kenntnis aller Befehle leicht zu handhaben, wovon die wichtigsten via Menu erreichbar sind.

- **Compiler:** Erklärt nebst der Arbeitsweise allfällige Besonderheiten zu anderen Modula-2 Compilern. Die systemspezifischen Unterschiede und insbesondere das Modul SYSTEM sind beschrieben.

- **Linker:** Erklärt, wie Modula-2 Programme auf dem Amiga ablauffähig gemacht werden.

- **Fehlerlister:** Zeigt die Möglichkeit, lesbare Fehlerprotokolle zu erhalten. Ist nur notwendig, wenn man m2emacs *nicht* verwendet.

- **Anmerkungen zu Amiga Modula-2:** Erklärt die Unterschiede zu anderen Implementationen von Modula-2

- **Laufzeitsystem:** Erklärt die Funktion des Laufzeitsystems.

- **Technische Angaben:** Gibt Informationen über interne Details von M2Amiga.

- **Programmierhinweise:** Diese Kapitel soll helfen in C geschriebene Amiga-Programme auf Modula-2 umzuschreiben.

- **Bibliotheken:** Zeigt die Definitionen der mitgelieferten Bibliotheken und erklärt kurz aber detailliert die Funktion der einzelnen Variablen und Prozeduren.

- **Schnittstellen-Moduln:** Die Definitionen aller Amiga-Libraries (und Devices). Die Definitionen sind exakte Übersetzungen der offiziellen C-Schnittstellen. Für die Funktionsweise sei darum auf die Amiga-Literatur verwiesen.

- **Cross-Reference:** Ein Index über alle mitgelieferten Moduln.

Notation

Neu einzuführende Begriffe erscheinen zuerst im **Fettdruck**, gefolgt von der Definition oder einem entsprechenden Verweis.

Begriffe, die sich auf einen aktuellen Bildschirminhalt beziehen, Eingaben oder Programmteile sind in einer `nichtproportionalen` Schrift dargestellt, manchmal auch durch Anführungszeichen hervorgehoben.

Anmerkung: Abschnitte, die mit dem Wort **Anmerkung** beginnen, geben weitere Informationen oder nützliche Hinweise zum Thema.

Warnung: Abschnitte, die mit dem Wort **Warnung** beginnen, sprechen schwerwiegendere Probleme des gegenwärtigen Themas an. Sie sind zusätzlich durch einen Rahmen hervorgehoben.

Beispiel. Beispiele sind mit dem Wort **Beispiel** eingeleitet. Zusätzlich sind sie durch einen Balken am linken Rand gekenzeichnet.

Terminologie

Folgende Bezeichnungen werden in diesem Handbuch verwendet:

Bildschirm soll hier nicht den Monitor benennen, sondern die der Amiga-Nomenklatur entsprechenden "Screens", also jene Einheiten, die durch ein bestimmtes Erscheinungsbild (Auflösung, Farbenzahl und Palette) ausgezeichnet sind.

CLI bezeichnet das sogenannte "Command Line Interface", übersetzt die Befehls-Zeilen-Schnittstelle, also die Benutzerschnittstelle des Amiga, welche die Eingabe von Programmnamen und Argumenten erlaubt.

Fenster ("Window") sind eine Art Arbeitsbereiche. Ein einzelnes Fenster wird behandelt, wie wenn man einen ganzen Computerschirm vor sich hätte. Fenster können ausserdem auf verschiedene Arten angeordnet werden, d.h in den Hintergrund gerückt, vergrössert, verkleinert oder überlappt werden.

Optionen sind zusätzliche Anweisungen an ein Programm. Diese werden oft auch Schalter genannt, da man gewisse Eigenschaften ein- oder ausschaltet. So ist es beispielsweise möglich, den Compiler anzuweisen, keine Statusinformationen auszugeben. Allerdings gibt es auch einige Optionen, die zusätzliche Angaben verlangen. Für diese ist das Wort Schalter eher irreführend.

Workbench zu Deutsch nichts anderes als der Arbeitstisch, bezeichnet die grafische Benutzerschnittstelle, die dank den Ikonen eine einfache Bedienung des Amiga ermöglicht.

<ALT> bezeichnet die ALT-Taste auf der Amiga-Tastatur. Die entsprechenden Bezeichnungen gelten ebenfalls für folgende Tasten: <BACKSPACE> (oder ← auf anderen wie US-Amiga 1000), <CTRL>, <DEL>, <ENTER>, <ESC>, <HELP>, <RETURN> (↵), <SHIFT> (⬆), <TAB> (→|), sowie für <F1> bis <F10> für die Funktionstasten.

<CTRL-A> bezeichnet die Tastenkombination von CTRL-Taste und der entsprechenden Buchstabentaste (hier "A") auf der Amiga-Tastatur. Solche Kombinationen sind üblich für <ALT>, <CTRL> und <SHIFT>. Für die Erzeugung drücke man zuerst die entsprechende Sondertaste und dann die Buchstabentaste.

^B ist eine Abkürzung für <CTRL-B>.

<→>,<←>,<↑>,<↓> bezeichnen die vier Pfeiltasten.

Verweise

An einigen Stellen wird zur Erläuterung auf die Amiga- und Modula–2 Dokumentation verwiesen. Die entsprechenden Publikationen sind:

[APM] Ingolf Krüger:
Amiga, Programmieren mit Modula-2
Markt & Technik

[DOS] Commodore Amiga Inc.:
The Amiga DOS Manual
2nd Edition
Bantam Computer Books

[HARDWARE] Commodore Business Machines, Inc.:
Amiga Hardware Reference Manual
Addison-Wesley

[INTUITION] Commodore Business Machines, Inc.:
Amiga Intuition Reference Manual
Addison-Wesley

[MC68000] Motorola Semiconductors:
MC68000 16/32-Bit Microprocessor
Programmer's Reference Manual

[PIM3] Niklaus Wirth:
Programming in Modula-2
Third edition
Springer

[RKM1] Commodore Business Machines, Inc.:
ROM Kernel Manual Part 1: Exec
Addison-Wesley

[RKM2] Commodore Business Machines, Inc.:
ROM Kernel Manual Part 2: Libraries and Devices
Addison-Wesley

2 Installation

Inhalt der Disketten 3

Dateien geringerer Bedeutung 5

Dateien mit fester Anordnung 7

Andere Dateien 7

Der erste Schritt 9

Installation auf Floppy Disk 9

Installation auf Festplatte 12

Wie geht es weiter? 12

Inhalt der Disketten

Kontrollieren Sie zunächst die Vollständigkeit des erworbenen Produkts. Stellen Sie dabei sicher, dass die Disketten schreibgeschützt sind, d.h. die Schalter auf den Disketten offen sind.

Die Diskette 1, die Systemdiskette, sollte folgende Dateien enthalten.

LiesMich	Enthält Änderungen zum Handbuch, die nicht mehr darin aufgenommen werden konnten.
Startup-Add	Textdatei mit den Ergänzungen zu der Datei Startup-Sequence.
Def.zoo	Archivdatei, die alle Definitionsmoduln von M2Amiga enthält.
M2	Verzeichnis, das alle zu M2Amiga gehörenden Dateien enthält.
M2/m2c	M2Amiga-Compiler.
M2/m2l	Linker (Binder).
M2/m2emacs	Bildschirmorientierter Editor. Dieser Editor ist speziell auf die Bedürfnisse des Modula-2-Programmierers angepasst.
M2/m2project	Programm zum Anlegen eines Projektverzeichnisses.
M2/m2error	Programm zum Erzeugen einer Fehlerliste. Dieser kommt nur zum Einsatz, wenn Sie m2emacs *nicht* einsetzen wollen.

M2/Fehler-Meldungen	Datei, welche die Fehlermeldungen in Klarform enthält. Diese werden von m2emacs und m2error benötigt.
M2/path	Pfaddatei.
M2/icons/txt.info	Standardikone, die vom Editor für neuerstellte Textdateien verwendet wird.
M2/icons/sym.info	Standardikone, die vom Compiler für neuerstellte Symboldateien verwendet wird.
M2/icons/obj.info	Standardikone, die vom Compiler für neuerstellte Objektdateien verwendet wird.
M2/icons/ref.info	Standardikone, die vom Compiler für neuerstellte Referenzdateien verwendet wird.
M2/icons/bin.info	Standardikone, die vom Linker für neuerstellte Programmdateien verwendet wird.
M2/icons/dir.info	Standardikone, die von m2project für das Projektverzeichnis verwendet wird.
M2/icons/txtDir.info	Standardikone, die von m2project für das Verzeichnis der Textdateien verwendet wird.
M2/icons/symDir.info	Standardikone, die von m2project für das Verzeichnis der Symboldateien verwendet wird.
M2/icons/objDir.info	Standardikone, die von m2project für das Verzeichnis der Objektdateien verwendet wird.

M2/icons/refDir.info	Standardikone, die von m2project für das Verzeichnis der Referenzdateien verwendet wird.
M2/icons/binDir.info	Standardikone, die von m2project für das Verzeichnis der Programmdateien verwendet wird.
M2/Modules	Verzeichnis, das alle mitgelieferte Bibliotheken und Schnittstellen-Moduln enthält.
M2/Modules/...	Die kompilierten Versionen der in "Bibliotheken" und "Schnittstellen-Moduln" beschriebenen Moduln.

Die Funktion dieser Dateien wird an anderer Stelle noch genauer erklärt. Vorerst ist nur von Bedeutung, welche Dateien überhaupt gebraucht werden. Ausserdem ist wichtig, ob diese Dateien an bestimmte Verzeichnisse gebunden sind, oder ob Sie irgendwo auf Ihrer Diskette oder Festplatte abgelegt werden dürfen.

Die Dateien der Diskette 2, der Demonstrationsdiskette, sind nicht festgelegt. Es handelt sich um eine Auswahl der bei Zusammenstellung interessantesten Demonstration-Programme. Jedes Programm ist dabei in einem Projektverzeichnis abgelegt. Erläuterungen sind ebenfalls innerhalb des Projekts und haben die Endung `.doc`.

Es ist übrigens keineswegs verboten, die Demonstrationsprogramme (Diskette 2) mit anderen Modula-2-Anwendern auszutauschen. Erst der rege Erfahrungsaustausch kann alle Möglichkeiten eines Produkts aufzeigen, was sehr wohl auch in unserem Interesse liegt.

Dateien geringerer Bedeutung:

Die Datei `LiesMich` enthält all die Informationen, die bei Druck des Handbuchs noch nicht bekannt waren. Diese Datei ist für den Betrieb von M2Amiga ohne Bedeutung.

Die Datei `Startup-Add` enthält Anweisungen, die in die Datei `Startup-Sequence` eingefügt werden sollen. Danach ist auch diese Datei nicht mehr nötig. Diese zusätzlichen Befehle sind:

```
Assign M2: SYS:
Stack 20000
```

Die Assign-Anweisung definiert ein logisches Gerät mit dem Namen `M2:`. Die Text- und Objektikonen enthalten als Defaulttool `M2:m2emacs` beziehungsweise `M2:m2l`. Die Pfad-Datei (`M2:path`) enthält den Eintrag `M2:Modules` als Verzeichnis, in welchem die Bibliotheken abgelegt sind.

Die Vergrösserung des Stapelbereichs ist für den Compiler erforderlich. Dieser arbeitet nach dem "Recursive-Descent"-Prinzip. Diese Technik erfordert eine grosse Zahl rekursiver Prozeduraufrufe, die einen grösseren Stapelbereich benötigen.

In der Datei `Def.zoo` sind alle Definitionsmoduln in Quellform abgelegt. Diese sind allerdings ausschliesslich zum Lesen zu verwenden und dürfen keinesfalls kompiliert werden. Anderenfalls können Sie die mitgelieferten Bibliotheksmoduln nicht mehr gebrauchen, da die Schlüssel unverträglich sind. Diese Datei ist eine Archivdatei, in der die Definitionsmoduln in komprimierter Form abgelegt sind. Sie können die Moduln mit dem auf der Demonstrationsdiskette mitgeliefertem, Programm `zoo` auspacken. Wechseln sie dazu mit dem CD-Befehl [DOS] in das Verzeichnis in dem die Definitionsmoduln abgelegt werden sollen. Geben sie dann den Befehl

```
zoo -extract Modula-2:Def.zoo
```

ein um alle Definitionsmoduln aus dem Archiv zu entnehmen. Sie können auch nur einzelne Module dem Archiv entnehmen, indem sie den Namen des Moduls angeben.

Beispiel: Sie wollen dem Archiv das Modul InOut.def entnehmen

```
zoo -extract Modula-2:Def.zoo InOut.def
```

Dateien mit fester Anordnung:

Die Dateien `Fehler-Meldungen`, `path` und das Verzeichnis `icons` werden im Verzeichnis `M2:` erwartet. Die Datei `Fehler-Meldungen` wird von den Programmen `m2emacs` und `m2error` benötigt. Sie enthält alle Fehlermeldungen in Textform, da in den vom Compiler erzeugten Fehlerdateien nur Fehlernummern enthalten sind.

Die Datei `path` enthält eine Liste aller Verzeichnisse, in denen sich zu importierende Moduln befinden. Die mitgelieferte `path`-Datei verweist nur auf das Verzeichnis `M2:Modules`.

Das Verzeichnis `icons` enthält die Ikonen für alle verschiedenen Datei- und Verzeichnistypen, die von den M2Amiga-Systemprogrammen erzeugt werden. Falls Ihnen die mitgelieferten Ikonen nicht zusagen, können Sie sie mit einem Ikoneneditor selbst modifizieren. Natürlich sind diese Dateien vor allem in der Workbench-Umgebung wichtig. Andernfalls können Sie diese Dateien auch löschen.

Andere Dateien:

Alle anderen Dateien können Sie (fast) beliebig plazieren. Sie müssen nur darauf achten, dass gewisse Dateien vom System oder vom Amiga-Betriebssystem automatisch gefunden werden müssen.

Dies ist einerseits der Fall bei den Programmen `m2emacs` und `m2l`. Die Ikonen für die Text- und Objektdateien enthalten als Defaulttool den Eintrag `M2:m2emacs` und `M2:m2l`. Folglich müssen `m2emacs` und `m2l` in einem gemeinsamen Verzeichnis liegen, dem sie mit dem Assign-Befehl (→[DOS]) den logischen Namen `M2:` zuordnen. Eine andere Möglichkeit ist, dass Sie in den Ikonen `M2:icons/txt.info` und `M2:icons/obj.info` mit Hilfe des Info-Befehls der Workbench das Defaulttool abändern.

Für die Bibliotheksmoduln gilt analoges. In der Datei `M2:path` muss vermerkt sein, wo diese zu finden sind. In der mitgelieferten `path`-Datei ist ein einziges Verzeichnis `M2:Modules` angegeben. Falls die

Moduln nicht dort abgelegt sind, muss die Pfad-Datei entsprechend geändert oder erweitert werden.

Die verbleibenden Programme `m2c`, `m2error` und `m2project` dürfen sich in einem beliebigen Verzeichnis befinden. Dieses Verzeichnis ist vorzugsweise im Pfad [DOS] vorhanden, um den Aufruf der Programme zu vereinfachen.

Die Dateien `m2c` und `m2l` werden auf jeden Fall benötigt. Das Programm `m2project` benötigen Sie nur, um Projektverzeichnisse zu erstellen. Von den Programmen `m2emacs` und `m2error` brauchen Sie normalerweise nur eines. Der mitgelieferte Editor `m2emacs` gibt die Fehlermeldungen im Klartext aus und setzt überdies den Cursor auf die Fehlerstelle. Falls Sie jedoch lieber mit einem anderen Editor arbeiten, dann benötigen Sie `m2error`. Der Compiler produziert nämlich nur eine codierte Fehlerdatei. Das Programm `m2error` erstellt aus Ihrem Quelltext und der Fehlerdatei eine Fehlerliste, in der die fehlerhaften Zeilen und Fehlermeldungen enthalten sind.

Damit Sie M2Amiga verwenden können, kopieren Sie die Dateien auf Ihre Festplatte. Falls Sie mit Disketten arbeiten, erstellen Sie eine Startdiskette, die neben den M2Amiga-Dateien noch einen Teil der auf der Workbench-Diskette benötigten Dateien enthält.

Bevor Sie sich diese Dateien ansehen oder gar mit der Installation beginnen, sollten Sie unbedingt eine Sicherheitskopie der M2Amiga-Diskette anfertigen. Dies können Sie mit dem Befehl

```
Diskcopy df0: to df1:
```

tun. Wenn Sie dazu aufgefordert werden, sollten Sie die mitgelieferte Diskette in das Laufwerk 0 (internes Laufwerk) und eine leere Diskette in das Laufwerk 1 (externes Laufwerk) legen. Achten Sie darauf, dass die Originaldiskette vor dem Überschreiben geschützt ist. Der Schreibschutz ist dann aktiv, wenn der Schreibschutzschieber der Diskette *offen* ist.

Nachdem Sie die Sicherheitskopie erstellt haben, sollten Sie nur mit dieser Kopie weiterarbeiten. Die Originaldiskette versorgen Sie am besten an einen sicheren Ort, also beispielsweise *nicht* auf dem Monitor

und auch *nicht* in Reichweite von Haustieren. Die Kopie wird am besten auch schreibgeschützt. Wenn in der nachfolgenden Anleitung von der Originaldiskette gesprochen wird, sollten Sie die soeben erstellte Kopie verwenden.

Diese Hinweise sollten bereits genügen, um sich Ihr System so zusammenzustellen , wie es für Ihre Konfiguration (Diskette/Festplatte, verfügbarer Speicher usw.) am besten ist. Um Ihnen die Arbeit etwas zu erleichtern, folgen nun als Beispiele die Installation für den Betrieb mit 2 Diskettenlaufwerken und für den Betrieb mit einer Festplatte.

Der erste Schritt

Bevor Sie mit der Installation beginnen, sollten Sie unbedingt die Datei mit dem Namen `LiesMich` anschauen. In dieser Datei sind allfällige Neuerungen, die nicht mehr ins Handbuch aufgenommen werden konnten, verzeichnet.

Arbeiten Sie mit dem CLI, so tippen Sie bitte:

```
Modula-2:M2/m2emacs Modula-2:LiesMich
```

bevorzugen Sie die Workbench, so doppelklicken Sie die Ikone mit dem Namen `LiesMich`.

Spätestens jetzt ist allerdings die Zeit für die Sicherheitskopie gekommen. Aber die haben Sie ja schon längstens gemacht, oder?

Installation auf Floppy Disk

Starten sie Ihren Amiga mit der Workbench Diskette, die Sie üblicherweise benutzen. Legen Sie eine leere Diskette in das externe Laufwerk (`df1:`) und formatieren Sie diese mit dem Befehl:

```
format drive df1: name M2Amiga
```

Kopieren Sie nun alle wichtigen Dateien von Ihrer Originaldiskette auf die soeben formatierte Diskette mit dem Befehl:

```
copy Modula-2:M2 M2Amiga: all +
```

Es ist wichtig, dass Sie das Pluszeichen (+) ebenfalls eingeben, und zwischen dem Pluszeichen und dem Drücken der <RETURN>-Taste keine Leerzeichen eingeben. Nun wird den Copy Befehl von der Workbench Diskette geladen, aber noch nicht ausgeführt. Wenn die LED des Laufwerks, in dem sich die Workbench-Diskette befindet, erloschen ist, entnehmen Sie die Workbench-Diskette und legen stattdessen die Originaldiskette ein. Drücken Sie danach nochmals <RETURN>, und die Dateien werden kopiert.

Wenn der Kopiervorgang abgeschlossen ist, können Sie die Originaldiskette wieder entfernen und die Workbench-Diskette wieder einlegen. Es wurden alle für den Gebrauch von M2Amiga relevanten Dateien auf die leere Diskette überspielt. Damit aus dieser Diskette eine "workbenchartige" Diskette wird, die Sie anstelle der Workbench zum Aufstarten des Amigas verwenden können, müssen noch einige Dateien kopiert werden. Sie sollten dazu die Verzeichnisse "L", "S" und "LIBS" von Ihrer Workbench-Diskette auf die neue Diskette kopieren.

```
MakeDir M2Amiga:L
Copy L: M2Amiga:L
MakeDir M2Amiga:S
Copy S: M2AMiga:S
MakeDir M2Amiga:LIBS
Copy LIBS: M2Amiga:LIBS
```

Weiterhin benötigen Sie die Befehle die in den Verzeichnissen `C:`, `System:` und `Utilities:` enthalten sind. Kopieren sie aber nur die Befehle, die Sie wirklich benötigen, den sonst haben sie zuwenig Platz auf der Diskette. Dasselbe gilt für das `DEVS:`-Verzeichnis. In diesem Verzeichnis sind neben den allgemeinen Gerätetreibern auch noch alle Drucker- und Tastaturtreiber enthalten. Kopieren Sie nur diejenigen Drucker- und Tastaturtreiber, die sie wirklich benötigen. Eine mögliche Auswahl wäre:

```
MakeDir M2Amiga:DEVS
Copy DEVS: M2Amiga:DEVS
MakeDir M2Amiga:C
Copy C:(Dir|Type|Copy|Delete) M2Amiga:C
Copy C:(Assign|Path|Run|Execute) M2Amiga:C
Copy C:(List|MakeDir|Rename|NewCLI) M2Amiga:C
Copy C:(If|Else|Endif|Echo) M2Amiga:C
Copy C:(Cd|Loadwb|EndCLI|Stack) M2Amiga:C
MakeDir M2Amiga:System
Copy DF0:System/(CLI|DiskCopy|Format)#? M2Amiga:System
```

Auf das FONTS:-Verzeichnis sollten Sie verzichten, denn dafür hat es kaum noch Platz auf der Diskette.

Im S: Verzeichnis befindet sich die Datei Startup-Sequence. In dieser Datei stehen alle Befehle, die beim Aufstarten des Systems ausgeführt werden. Zu den bereits in der Startup-Sequence existierenden Befehlen sollten noch diese beiden hinzugefügt werden:

```
Stack 20000
Assign M2: SYS:
```

Sie können diese zwei Zeilen mit einem Editor in die Datei Startup-Sequence einfügen. Vergessen Sie auch auf keinen Fall, den richtigen Tastaturtreiber zu aktivieren, beispielsweise durch den Befehl

```
SetMap d
```

in der Datei Startup-Sequence. (Hier wurde als Beispiel der deutsche Tastaturtreiber geladen).

Zum Schluss müssen Sie noch dafür sorgen, dass der Amiga die neue Diskette auch als Bootdiskette erkennt. Dies wird mit dem Install-Befehl gemacht:

```
Install drive df1:
```

Nun ist die Installation abgeschlossen. Sie können mit der soeben erstellten Diskette den Amiga neu aufstarten (<CTRL>-<Amiga>-<Amiga>). Es empfiehlt sich jedoch, zuvor auch von dieser Diskette eine Sicherheitskopie anzufertigen. Damit ersparen Sie sich, falls die Diskette zerstört wird, die zeitaufwendige Installation ein weiteres Mal durchführen zu müssen.

Installation auf Festplatte

Nachdem sie Ihren Amiga aufgestartet haben, legen Sie die Originaldiskette in das interne Laufwerk (df0:). Legen sie nun für M2Amiga ein eigenes Verzeichnis auf Ihrer Festplatte an und kopieren Sie alle wichtigen Dateien in dieses Verzeichnis. In den nun folgenden Beispielen wird angenommen, dass Ihre Festplatte mit `dh0:` angesprochen wird, und dass Sie darauf ein Verzeichnis mit dem Namen `M2Amiga` erstellen:

```
MakeDir dh0:M2Amiga
copy df0:M2 dh0:M2Amiga: all
```

Folgende zwei Zeilen müssen in die Datei `Startup-Sequence` eingefügt werden:

```
Stack 20000
Assign M2: dh0:M2Amiga
```

Sie können diese zwei Zeilen mit einem Editor einfügen.

Nun ist die Installation abgeschlossen. Sie sollten den Amiga neu aufstarten, damit die in der Datei `Startup-Sequence` eingefügten Befehle auch ausgeführt werden.

Wie geht es weiter?

Sie sollten jetzt ein ordnungsgemäss installiertes System besitzen. Wenn Sie weiterlesen, sollten Sie am besten vor dem Computer sitzen und die erwähnten Dinge gleich ausprobieren. Ist etwas nach dem Lesen des Handbuchs unklar, hilft meist die Praxis weiter. Suchen Sie die

erwähnten Programme und "bewegen" Sie sich etwas im System, und Sie werden sich bald wie zu Hause fühlen!

3 Arbeitsumgebung Amiga Modula-2

Übersicht 3

Programmaufruf 3

Optionen 4

Argumente 4

Hilfe 5

Das Projekt-Konzept 6

Einleitung 6

Details zum Projekt 7

Zusammenspiel verschiedener Projekte 9

m2project 10

Dieser Teil zeigt, aus welchen Komponenten Amiga Modula-2 besteht, wie diese anzuwenden sind und von welchen Voraussetzungen die Programme ausgehen.

Übersicht

Die Grundversion von Amiga Modula-2 besteht aus fünf Programmen. Diese fünf Programme sind:

m2c	Der Amiga Modula-2 Compiler
m2l	Der Linker, der aus den vom Compiler erzeugten Dateien ablauffähige Programme macht.
m2emacs	Der bildschirmorientierte Editor, der alle vom Compiler gefundenen Fehler der Reihe nach anzeigt.
m2project	Ein Hilfsprogramm, das die Verwaltung der einzelnen Dateien vereinfacht.
m2error	Ein Hilfsprogramm, das erst zur Anwendung kommt, falls man einen anderen Editor benützen möchte.

Das Programm "m2project" ist in diesem Teil beschrieben, die anderen Programme sind in einem eigenen Teil erläutert. Allgemeingültige Angaben werden im unmittelbar folgenden Abschnitt "Programmaufruf" gegeben.

Programmaufruf

Alle Programme des Amiga Modula-2 Systems sind mit einer einheitlichen Benutzerschnittstelle ausgestattet. Wenn Sie den Aufrufmechanismus eines Programms kennen, dann wissen Sie auch, wie die anderen Programme zu aktivieren sind. Falls Sie diese Schnittstelle schätzen, empfehlen wir Ihnen, alle eigenen Programme mit der gleichen Schnittstelle zu versehen. Verwenden Sie dazu das im Teil "Bibliotheken" beschriebene Modul "Arguments".

Die Arbeitsweise der Programme kann zusätzlich mit Optionen (Schalter) beeinflusst werden. Optionen können nur vom CLI aus angegeben werden. Ein Programm funktioniert aber immer auch ohne spezielle Optionen, diese bieten lediglich zusätzliche Möglichkeiten.

Optionen

sind durch einzelne Buchstaben bezeichnet und werden mit einem "-" eingeleitet. Optionen müssen vor den restlichen Programmargumenten stehen. Parameterlose Optionen können wie folgt angegeben werden:

`befehl -x`	setzt die Option "x".
`befehl -xl`	setzt die Optionen "x" und "l".
`befehl -x -l`	setzt die Optionen "x" und "l".

Optionen mit Parameter werden folgendermassen angegeben:

`befehl -d Verzeichnis`	setzt die Option "d" mit dem Argument "Verzeichnis".

Wird das Programm von der Workbench aus aufgerufen, sind gewisse Optionen automatisch gesetzt (beispielsweise Ikonenerzeugung).

Argumente

werden auf der Befehlszeile hinter den Optionen angegeben und sind durch Leerzeichen voneinander getrennt; allerdings können auch Leerschläge in ein Argument eingefügt werden. Details entnehme man dem Kapitel "Arguments" im Teil "Bibliotheken".

In der Workbench-Umgebung können die Argumente durch "erweiterte Selektion" übergeben werden. Dazu wähle ("klicke") man das auszuführende Programm, gefolgt von allen gewünschten Argumenten. Beim letzten Argument muss entweder "doppelgeklickt", oder die Funktion "Open" im Workbench-Menu gewählt werden, um das Ende der Eingabe anzugeben. Dabei muss für jede Auswahl die <SHIFT>-Taste gedrückt werden, da sonst nur eine Selektion möglich ist.

Ein Programm kann allerdings auch ohne Argumente aufgestartet werden. Das Programm wechselt in einen interaktiven Modus, d.h. es fragt solange nach einem Argument, bis ein leeres Argument eingegeben wird. Diese Lösung erlaubt das Erstellen besonders einfacher Befehlsfolgen, die ohne weiteres Zutun ablaufen. Um zehn Programme zu compilieren, braucht es nur eine Datei mit den entsprechenden Dateinamen, und der Befehl

```
m2c <datei
```

compiliert alle in "datei" vermerkten Moduln.

Hilfe

ist jederzeit zu bekommen. Erscheint ein einzelnes "?" vor dem ersten Programmargument, wird der Befehl *nicht* ausgeführt. Stattdessen erscheint eine Meldung, die etwa so aussieht:

```
1> m2error ?
m2error, 3.2, 1.11.88, © AMSoft
Aufruf: m2error [-x] {Dateiname}
```

Die erste Zeile bezeichnet den Programmnamen, Versionsnummer und Datum. Die zweite Zeile beschreibt die Anwendung. Alles in eckigen Klammern ([-x]), kann weggelassen werden. Alles in geschweiften Klammern ({Dateiname}) bezeichnet die beliebige Wiederholung (auch null Mal) dieses Argumenttyps. Diese Zeile besagt also, dass "m2error" mit einer Option, die weggelassen, und mit keinem, einem oder beliebig vielen Modul- oder Dateinamen aufgerufen werden kann. Sind mehrere Buchstaben hinter dem Strich, der die Optionen einleitet, so ist das für jede Option einzeln zu verstehen:

```
befehl [-bim]
```

heisst, dass das Programm mit keiner, einer, zwei oder allen drei Optionen aufgerufen werden kann.

Das Projekt-Konzept

Einleitung

Im Allgemeinen wird ein grösseres Modula-2 Programm aus mehreren Moduln bestehen. Jedes Modul umfasst seinerseits mehrere Dateien, wie zum Beispiel die Textdateien (das Definitions-, Implementations- oder Programm-Modul), die Symboldateien, die Referenzdateien und zu guter letzt die fertig gelinkten Amiga Programme.

Entwickelt man nun diverse Modula-2 Programme, hat man schnell eine grössere Anzahl Dateien, die - ohne besondere Vorkehrungen - unübersichtlich auf der Diskette oder Harddisk "herumliegen". Man verliert somit schnell die Übersicht, welche Files zu welchem Modul gehören, und welche Moduln zu welchem Modula-2 Programm gehören.

Das Betriebssystem bietet uns schon eine Möglichkeit an, um auf der Diskette oder Harddisk eine übersichtliche Anordnung mehrerer Dateien vornehmen zu können; die Verzeichnisse bzw. Unterverzeichnisse (directories oder drawers). Wir benützen diese Möglichkeit und bauen sie zu einem eigentlichen Konzept aus; unser Projekt-Konzept. Es genügt nämlich nicht, wenn sie als Benutzer einfach Ihre Ordnung auf dem externen Speicher vornehmen, unsere Programme (Compiler, Linker, Loader, Debugger...) Ihre Anordnung der Dateien aber unberücksichtigt lässt.

Wenn sie ein Modula-2 Programm entwickeln - nehmen wir als Beispiel gerade den Compiler - kann man dieses Vorhaben als Projekt bezeichnen. Zu diesem Projekt gehören diverse Moduln. Um bei unserem Beispiel des Compilers zu bleiben, gibt es sicher ein Modul das den Text einliest und die Syntax prüft (Scanner), ein Modul das die Semantik überprüft (Parser) und ein drittes Modul erzeugt schliesslich den Code (Generator). All diese Dateien gehören logisch zusammen und werden demzufolge in einem Verzeichnis zusammengefasst. Somit kann man folgendes festhalten:

"Ein Projekt entspricht einem Verzeichnis"

Details zum Projekt

Bis jetzt ist noch nichts wesentlich Spektakuläres an unserem Konzept zu erkennen. Es liegt auf der Hand, alle zusammengehörenden Moduln in einem eigenen Verzeichnis abzulegen! Erinnern wir uns aber an den ersten Abschnitt der Einleitung. Dort haben wir gesehen, dass es viele verschiedene Typen von Dateien gibt. Sie unterscheiden sich durch spezifische Endungen:

.def	Textdatei eines Definitions-Moduls
.mod	Textdatei eines Implementations- oder Programm-Moduls
.sym	übersetztes Definitions-Modul
.obj	übersetztes Implementations- oder Programm-Modul
.ref	Datei, die die Debuggerinformation enthält
---	File des gelinkten Programms hat keine Endung

Schaut man sich ein Projektverzeichnis mit dem "dir"-Befehl des Betriebssystems an, stehen aber nicht alle Dateien gleichen Typs beieinander, sondern man findet alle Dateien, die zu einem Modul gehören, alphabetisch aufgelistet. In unserem Beispiel von vorhin sieht das Projektverzeichnis folgendermassen aus:

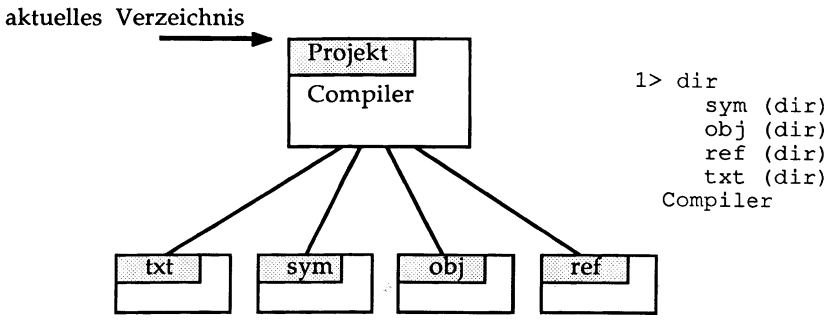
```
1> dir
  Compiler          Compiler.mod
  Generator.def     Generator.mod
  Generator.obj     Generator.ref
  Generator.sym     Parser.def
  Parser.mod        Parser.obj
  Parser.ref        Parser.sym
  Scanner.def       Scanner.mod
  Scanner.obj       Scanner.ref
  Scanner.sym
```

Die Liste ist zwar alphabetisch, aber nur schon diese drei Moduln ergeben eine recht lange Liste. Manchmal wäre es vielleicht besser, wenn man alle Dateien des gleichen Typs beieinander hätte. Um dies zu erreichen, kann man das Projektverzeichnis mit Unterverzeichnissen weiter unterteilen. Unser Projektkonzept erkennt Unterverzeichnisse mit folgenden Namen:

bin	Dieses Unterverzeichnis enthält alle fertig gelinkten Amiga Programme.
obj	Wie der Name schon sagt enthält dieses Unterverzeichnis alle ".obj"-Dateien.
ref	Hier drin finden sich alle ".ref"-Files.
sym	Hier sind alle ".sym"-Files zu finden.
txt	Und hier sind alle Programmtexte abgespeichert, d.h. die ".def"- und ".mod"-Dateien

Wichtig ist, dass es NICHT obligatorisch ist, mit diesen Unterverzeichnissen zu arbeiten. Man kann sie einrichten, wenn man will; man kann aber auch alle oder einen Teil davon weglassen. Wenn man aber ein Unterverzeichnis eingerichtet hat, muss man es benutzen. Das heisst, dass bei eingerichtetem "txt" Unterverzeichnis alle Programmtexte in diesem abgelegt werden müssen. Mit anderen Worten: Existiert ein "txt" Unterverzeichnis, so werden Texte die direkt im Projektverzeichnis abgelegt sind, von allen M2Amiga Programmen ignoriert.

Unser obiges Beispielprojekt könnte also folgendermassen aussehen:



Beispiel eines Projekts

Zur Anmerkung: Das "bin" Unterverzeichnis wurde ausgelassen, so dass sich das fertig gelinkte Programm "Compiler" direkt im Projektverzeichnis befindet.

Zusammenspiel verschiedener Projekte: die Suchstrategie

Es können aber auch Dateien aus anderen Projekten importiert werden. Dafür gibt es die Pfaddateien. Pfaddateien sind normale Textdateien, die auf jeder Zeile einen Projektnamen angeben. Es gibt eine lokale Pfaddatei, sowie eine globale Pfaddatei. Der Name einer Pfaddatei ist "path". Die lokale Pfaddatei ist im Projektverzeichnis, die globale befindet sich im "M2:"-Verzeichnis. Die Suche geht folgendermassen vor: Zuerst wird das aktuelle Projekt, danach die Projekte der lokalen Pfaddatei und schliesslich die Projekte der globalen Pfaddatei durchsucht.

Zu beachten ist, dass in den Pfaddateien nicht etwa Unterverzeichnisse aufgezählt werden, sondern Projekte. In diesen Projekten gelten dieselben Suchregeln bezüglich der Unterverzeichnisse (txt, sym, obj, ref, bin) wie im aktuellen Projekt.

Meistens genügt die globale Pfaddatei, und man muss keine lokalen Pfaddateien definieren. Ein Beispiel einer globalen Pfaddatei:

```
M2:Modules
Arbeits-Disk:Library
Arbeits-Disk:AnderesProjekt
```

Wird eine benötigte Symboldatei im aktuellen Projekt nicht gefunden, wird zuerst im Verzeichnis "M2:Interfaces" nachgeschaut. Ist dort ein Unterverzeichnis "sym" vorhanden, wird in diesem gesucht, sonst im Verzeichnis selbst. Wurde die Datei nicht gefunden, geht die Suche bei den beiden folgenden Verzeichnissen analog weiter. Erst wenn sie auch dort nicht vorhanden ist, wird eine Fehlermeldung ausgegeben.

m2project

Das Programm "m2project" legt ein Projekt samt den gewünschten Unterverzeichnissen an. Um einzelne Unterverzeichnisse *nicht* erzeugen zu lassen, existieren die Optionen -o, -r, -b, -s, -t und -i:

-o	Erzeuge *kein* Unterverzeichnis "obj"
-r	Erzeuge *kein* Unterverzeichnis "ref"
-b	Erzeuge *kein* Unterverzeichnis "bin"
-s	Erzeuge *kein* Unterverzeichnis "sym"
-t	Erzeuge *kein* Unterverzeichnis "txt"
-i	Erzeuge *keine* Ikonen für die Verzeichnisse

Der Aufruf "`m2project -o IcePack`" erzeugt das Verzeichnis "IcePack" mit den Unterverzeichnissen "sym", "ref", "txt" und "bin"; alle Unterverzeichnisse werden mit einer Ikone versehen..

Ist bereits ein Verzeichnis mit dem angegebenen Namen vorhanden, wird *kein* neuesProjekt erzeugt.

4 Der Editor

Aufruf 3

Charakteristik des Editors 3

Spezielle Terminologie 3

Grundlegendes 5

Die Maus als Eingabegerät 6

Lesen und Abspeichern von Dateien 7

Löschen und Wiedereinsetzen von Textbereichen 9

Suchen und Ersetzen eines Textes 12

Ändern eines Textbereichs 13

Schalterstellungen 14

Bildschirminhalt auswählen und Fensterbehandlung 15

Makrobefehle 16

Modula-2 Befehle 17

Verschiedene Befehle 18

Betriebssystemumgebung 20

Optionen beim Aufruf 20

Rückgabewerte 21

Ein Beispiel 21

Befehlszusammenfassung 23

Aufruf

```
m2emacs [-d Verzeichnis][-bi] [Dateiname]
```

Charakteristik des Editors

m2emacs ist ein bildschirmorientierter Zeileneditor. Ein damit bearbeiteter Text besteht aus einer Anzahl von verschiedenen Textzeilen mit einer Länge bis zu 255 Zeichen. Jedes Zeichen kann sich über den gesamten 8-Bit Bereich erstrecken, beinhaltet insbesondere alle nationalen Sonderzeichen, wie in [DOS] vermerkt.

Auf jeder Zeile sind maximal 80 Zeichen sichtbar. Beinhaltet die Zeile mehr als 80 Zeichen, erscheint auf der 80. Position ein "$"-Zeichen. Die Anzahl der angezeigten Zeilen ist abhängig von der Zeilenzahl des Bildschirms (PAL/NTSC).

m2emacs bietet die Möglichkeit, mehrere Dateien gleichzeitig zu bearbeiten. Die Anzahl der Dateien ist nur durch die Grösse des Hauptspeichers begrenzt, denn m2emacs bearbeitet alle Dateien im Hauptspeicher.

m2emacs unterstützt auch die Maus als Eingabegerät, und zwar zum Setzen des Cursors wie auch zur Menuauswahl. Weitere Möglichkeiten finden sich im Kapitel "Die Maus als Eingabegerät".

Spezielle Terminologie

Textspeicher bezeichnet einen Speicherbereich, der von m2emacs verwaltet wird. Es existiert immer mindestens ein Textspeicher, der null oder mehr Zeichen enthält.

Cursor — bezeichnet die Stelle im Text, an der Änderungen stattfinden. Die Position des Cursors kann beliebig im Text verändert werden. Der Cursor ist mit einem roten (oder der Preferences-Einstellung Farbe 3 entsprechenden) Rechteck gekennzeichnet.

Marke — ist wie der Cursor eine ausgezeichnete Stelle im Textspeicher. Sie kann überall und jederzeit gesetzt werden. Die Marke ist nicht sichtbar, aber deren Position wird vom m2emacs vermerkt.

Bereich — bezeichnet den Textteil zwischen Cursor und Marke. Einige Operationen wirken direkt auf den Bereich.

Fenster — in m2emacs entsprechen nicht den Fenstern, wie sie von Intuition verwaltet werden. Stattdessen erlaubt m2emacs eine horizontale Unterteilung des Bildschirms, wobei jedes dieser Fenster das Bearbeiten einer einzelnen Datei ermöglicht.

Textname — ist einer von zwei Namen, die einem Textspeicher zugeordnet sind. Der erste dieser zwei ist der Textname, mit dem ein Textspeicher identifiziert wird und angewählt werden kann. Der zweite Name bezieht sich auf die Datei, die gerade im Textspeicher bearbeitet wird.

Der erste Textspeicher hat als Textnamen den Dateinamen ohne Pfad, beziehungsweise "main" (engl. Haupt-) falls keine Datei angegeben wurde. Der zugehörige Dateiname entspricht entweder dem Namen der Datei, die beim Aufruf von m2emacs als Parameter angegeben wurde, oder kann nachträglich auch noch durch einige Befehle verändert werden.

Statuszeile ist die unterste Zeile des Bildschirms. Alle Editorbefehle, die zusätzliche Parameter benötigen, fragen diese auf der Statuszeile ab. Bei Eingaben auf der Statuszeile kann übrigens nicht nur das letzte Zeichen mit <BACKSPACE>, sondern die ganze Eingabezeile mit <CTRL-X> gelöscht werden. Wurde der Befehl irrtümlich aufgerufen, kann er mit <CTRL-G> ganz abgebrochen werden.

Die Statuszeile dient ebenso der Anzeige von Fehlermeldungen oder sonstigen Informationen.

Grundlegendes

m2emacs kennt zwei verschiedene Operations-Modi, den Normalmodus und den Befehlsmodus. m2emacs befindet sich immer dann im Normalmodus, wenn kein Befehl ausgeführt wird. Nachdem ein Befehl abgeschlossen ist, kehrt m2emacs wieder in den Normalmodus zurück.

Der Normalmodus zeichnet sich folgendermassen aus:

- Zeichen, die in den Text eingefügt werden sollen, brauchen nur getippt zu werden. Der Normalmodus ist immer auch "Einfügemodus". Eingefügt werden können alle Zeichen, also auch das "Neue-Zeilen"-Zeichen (<RETURN>), das einen Zeilenumbruch bewirkt.

- Das Zeichen unter dem Cursor kann mit der <DEL>-Taste, dasjenige Zeichen unmittelbar vor dem Cursor mit der <BACKSPACE>-Taste gelöscht werden. Alle nachfolgenden Zeichen rücken entsprechend nach.

- Der Cursor kann mit den Pfeiltasten buchstaben- beziehungsweise zeilenweise bewegt werden. Der Cursor kann weder hinter das Zeilenende bewegt werden, noch ist es möglich, den Cursor über die letzte Zeile des Textes hinaus zu positionieren.

- Der Cursor kann auch schnell und einfach über weitere Strecken verschoben werden. So bewirkt das Drücken der <ALT>-Taste zusätzlich zu einer Pfeiltaste das Verschieben an den Anfang oder Ende eines Wortes (<ALT-←> und <ALT-→>), beziehungsweise das Blättern um eine Seite vor- oder rückwärts (<ALT-↓> und <ALT-↑>). Die <SHIFT>-Taste vergrössert den Schritt auf den Anfang oder das Ende der Zeile (<SHIFT-←> und <SHIFT-→>), beziehungsweise an den Anfang oder das Ende des Textspeichers (<SHIFT-↑> und <SHIFT-↓>).

- Als letzte Möglichkeit der Cursor-Positionierung kann auch die Maus benutzt werden. Dazu muss der Mauszeiger an die gewünschte Stelle gebracht, und der *linke* Mausknopf gedrückt werden.

- Verschiedene weitere Funktionen können entweder durch Tastenkombinationen (Kontroll und Escape-Sequenzen) oder durch Menuauswahl ausgeführt werden. Diese Funktionen sind im folgenden ausführlich beschrieben.

Der Befehlsmodus wird durch viele dieser weiteren Funktionen gestartet. Im Befehlsmodus springt der Cursor auf die unterste Zeile des Bildschirms und verlangt die Eingabe weiterer Informationen. Die Art der benötigten Information ist für jede Funktion erklärt. Ist der Befehl beendet, wird wieder in den Normalmodus zurückgekehrt.

Die unterste Zeile jedes Fensters ist die Fensterbegrenzungszeile. Auch auf der Begrenzungszeile sind Statusinformationen zu finden, allerdings haben diese nur für den in diesem Fenster sichtbaren Textspeicher Bedeutung. Die Begrenzungszeile besteht aus einer Zeile von Minussymbolen ("-"), durchbrochen vom Programmnamen "m2emacs". Daneben ist der Textname zu finden und schliesslich ist der dem Textspeicher zugeordnete Dateiname vermerkt. In der zweiten Spalte dieser Zeile wird auch markiert, ob der Textspeicher verändert wurde. Ein Stern ("*") bedeutet einen veränderten Textspeicher, das ursprüngliche Minuszeichen heisst, dass Datei und Textspeicher (noch) identisch sind.

Die Maus als Eingabegerät

Setze Cursor (linker Mausknopf)

kann statt mit den Pfeiltasten auch schneller mit der Maus ausgeführt werden. Um an eine gewünschte Textstelle zu gelangen, kann man mit der Maus den Zeiger an diese Stelle bewegen, und dort den *linken* Mausknopf drücken. Dies hat den zusätzlichen Effekt, dass das entsprechende Fenster aktiv wird.

Setze Marke (rechter Mausknopf)

Die Marke kann ebenfalls mit der Maus gesetzt werden. Ist der Mauszeiger an der gewünschten Markenposition, muss der *rechte* Mausknopf gedrückt werden. Gleichzeitig wird mit diesem Befehl auch der Cursor an diese Stelle gesetzt.

Blättern (linker / rechter Mausknopf)

Drückt man den linken, beziehungsweise den rechten Mausknopf in der Fensterbegrenzungszeile, blättert m2emacs den Text eine Seite vorwärts oder zurück.

Ausschneiden (linker / rechter Mausknopf)

eines Bereichs kann ähnlich einfach auch mit der Maus alleine ausgeführt werden. Die genaue Vorgehensweise schaue man bitte im nachfolgenden Abschnitt "Ausschneiden" nach.

Lesen und Abspeichern von Dateien

Öffne Datei (Projekt-Menu) / **<CTRL-X><CTRL-R>** / **<F1>**

liest den Inhalt einer Datei in den aktuellen Textspeicher. Ist dieser leer, springt der Cursor auf die Statuszeile und verlangt einen Dateinamen, der auch einen Pfad beinhalten darf. Wurde der Textspeicher verändert

– gekennzeichnet durch den Stern auf der Fensterbegrenzungszeile – versichert sich m2emacs durch die Frage "Änderungen vergessen [j/n]?" ob der ungesicherte Textspeicher wirklich überschrieben werden soll. Die Antwort "j" (abzuschliessen mit <RETURN>) überschreibt den Textspeicher, "n" ignoriert den Aufruf des Öffnen-Befehls. Ist der Textspeicher nicht leer aber unverändert, so gilt der Textspeicher als gesichert und wird durch den Inhalt der neu einzulesenden Datei überschrieben.

Das Löschen eines alten Textspeichers kann übrigens bis zu einigen Sekunden dauern.

Soll keine neue Datei eingelesen werden, genügt es, ohne Angabe eines Dateinamens die Taste <RETURN> zu drücken, und m2emacs kehrt in den Normalmodus zurück.

Lies Datei (Projekt-Menu) / **<CTRL-X><CTRL-V>**

Dieser Befehl ruft einen neuen Textspeicher ins Leben. Dieser Befehl ist gleichbedeutend mit: "Ich möchte eine andere Datei bearbeiten, während ich diese Datei bearbeite". Dieser Befehl ist vor allem nützlich, um aus verschiedenen Programmen Teile auszusondern, oder um Definitionsmoduln zu konsultieren.

Falls der eingegebene Dateiname nicht bereits einem Textspeicher zugeordnet ist und die Datei existiert, wird ein neuer Textspeicher kreiert und sofort auf dem Bildschirm angezeigt. Der alte Textspeicher ist noch zugreifbar und kann jederzeit angewählt werden. Ist die angegebene Datei bereits geöffnet oder gelesen, wird kein neuer Textspeicher kreiert, sondern nur auf den Textspeicher, der dieser Datei zugeordnet ist, umgeschaltet.

Eine Liste aller Textspeicher kann angezeigt werden. Dieser Befehl ist im letzten Abschnitt beschrieben.

Sichern (Projekt-Menu) / **<CTRL-X><CTRL-S>** / **<SHIFT-F1>**

schreibt den Inhalt des aktuellen Textspeichers auf diejenige Datei, die durch den dem Textspeicher zugehörigen Namen identifiziert ist. Ist dem Textspeicher kein Dateiname zugeordnet, kann die Datei nicht geschrieben werden. Der Dateiname wird durch das Öffnen oder das Umbenennen dem Textspeicher zugeordnet.

Konnte die Sicherung erfolgreich abgeschlossen werden, erscheint auf der Statuszeile die Anzahl der geschriebenen Zeilen.

Sichern als (Projekt-Menu) / **<CTRL-X><CTRL-W>**

schreibt den Inhalt des aktuellen Textspeichers auf eine Datei mit frei wählbarem Namen. Dieser Dateiname wird neu dem Textspeicher zugeordnet. Für das erneute Abspeichern genügt somit der Sichern-Befehl ohne Dateiname. Der Textspeicher ist wie beim Sichern nicht mehr länger als geändert markiert, und der Stern auf der Fensterbegrenzungszeile verschwindet.

Neuer Dateiname (Projekt-Menu) / **<CTRL-X><CTRL-F>**

ändert den Dateinamen des aktuellen Textspeichers auf einen neuen Namen. Die Angabe eines leeren Namens ist zwar möglich, meist aber unsinnig. Ohne gültigen Dateinamen kann die Datei nicht abgespeichert werden.

Sichern & Ende (Projekt-Menu) / **<CTRL-Z>**

schreibt den Inhalt des aktuellen Textspeichers auf die zugeordnete Datei und verlässt m2emacs. Existieren weitere veränderte Textspeicher, wird gefragt, ob m2emacs wirklich verlassen werden soll.

Ende (Projekt-Menu) / **<CTRL-X><CTRL-C>**

verlässt m2emacs. Falls aber veränderte Textspeicher existieren sollten, fragt m2emacs zur Sicherheit, ob man den Editor wirklich verlassen

will. Drückt man "j" ist die getane Arbeit endgültig verloren, bricht man den Ende-Befehl aber mit "n" ab, befindet man sich wieder im Normalmodus des Editors. Hier kann dann "Sichern" aufgerufen werden.

Lösche Textspeicher <CTRL-X>K

löscht einen Textspeicher aus dem Speicher. Wenn ein Textspeicher nicht mehr benötigt wird, sollte er gelöscht werden, damit der von ihm belegte Speicherbereich wieder freigegeben werden kann. Ein Textspeicher kann nur gelöscht werden, wenn er nicht am Bildschirm sichtbar ist. Ein unveränderter Textspeicher wird dann sofort gelöscht, sonst muss der Benutzer das Löschen zur Sicherheit nochmals bestätigen.

Löschen und Wiedereinsetzen von Textbereichen

Buchstabenweises Löschen funktioniert wie bereits beschrieben mit den Tasten <DEL> und <BACKSPACE>. Daneben existieren auch Möglichkeiten des wort- und zeilenweisen Löschens. Als Wort gilt dabei jede Zeichenkette bestehend aus alphanumerischen Zeichen von "A"-"Z", "a"-"z", "0"-"9" und allen Zeichen, deren Ordinalwert grösser oder gleich 128 ist (höchstes Bit gesetzt). Die Umlaute sind demnach den normalen Buchstaben gleichgesetzt.

Lösche Wort zurück <ESC><BACKSPACE> / <ALT-BACKSPACE>

löscht das Wort vor dem Cursor, falls sich dieser zwischen zwei Wörtern befindet. Ist der Cursor mitten in einem Wort, werden alle Zeichen dieses Worts gelöscht, die sich vor dem Cursor befinden.

Lösche Wort <ESC>D / <ALT-DEL>

löscht das Wort nach dem Cursor, falls sich dieser zwischen zwei Wörtern befindet. Ist der Cursor mitten in einem Wort, wird das Zeichen unmittelbar unter dem Cursor, sowie alle nachfolgenden Zeichen des Worts gelöscht.

Lösche Zeile <CTRL-X><CTRL-D> / <CTRL-X><DEL> / <F10>

löscht die Zeile, auf der sich der Cursor gerade befindet. Diese Zeile wird in den sogenannten Zwischenspeicher übertragen und kann dort solange abgerufen werden, wie er nicht durch ein erneutes Löschen überschrieben wird. Für das Zurückholen des Zwischenspeichers schaue man unter "Einsetzen" nach.

Lösche bis Ende Zeile <CTRL-K> / <SHIFT-DEL>

löscht alle Zeichen dieser Zeile, die sich hinter dem Cursor befinden, einschliesslich des unter dem Cursor liegenden Zeichens. Ist die Zeile leer, d.h. befindet sich nur das Zeilenvorschubzeichen (line feed) auf ihr, wird dieses gelöscht. Der gelöschte Text wird in den Zwischenspeicher übertragen.

Lösche von Beginn der Zeile <SHIFT-BACKSPACE>

löscht alle Zeichen dieser Zeile, die sich vor dem Cursor befinden. Der gelöschte Text wird in den Zwischenspeicher übertragen.

Füge Zeilen zusammen <CTRL-X><CTRL-J>

fügt die unterliegende Zeile an die aktuelle Zeile an, d.h entfernt das Zeilenvorschubzeichen am Ende der Zeile. Die Cursorposition bleibt erhalten.

Setze Marke (Mausknopf rechts) **<CTRL-@> / <ESC>.**

setzt die Marke an der momentanen Cursorposition. Wird die Marke mit dem rechten Mausknopf gesetzt, muss der Mauszeiger an der gewünschten Stelle des Texts sein. Der Cursor springt darauf an die Stelle der Marke. In jedem Fall erscheint in der Statuszeile die Meldung `"[Marke gesetzt]"`.

Ausschneiden (Edit Menu) / **<CTRL-W>** / (Maus)

löscht den Bereich, d.h. den Text, der sich zwischen Marke (einschliesslich) und Cursorposition befindet, und überträgt den gelöschten Text in den Zwischenspeicher.

Dieser Befehl kann auch nur mit der Maus ausgeführt werden. Dazu drücke man den linken Mausknopf an der gewünschten Cursorposition, halte ihn gedrückt und drücke gleichzeitig den rechten Mausknopf, den man sofort wieder loslässt. Sobald auch der linke Knopf wieder losgelassen wird, ist der Bereich gelöscht und in den Zwischenspeicher kopiert.

Kopieren (Edit Menu) / **<ESC>W**

überträgt den Bereich in den Zwischenspeicher, lässt den Textspeicher aber unverändert. Es ist eine Abkürzung für das Ausschneiden und sofortige Wiedereinsetzen eines Bereichs.

Einsetzen (Edit Menu) / **<CTRL-Y>**

setzt den im Zwischenspeicher enthaltenen Text an der aktuellen Cursorposition ein. Der Zwischenspeicher bleibt bestehen und wird nicht verändert. Der Cursor steht neu unmittelbar hinter dem letzten Zeichen des eingesetzten Blocks; die Position der Marke bleibt indes unverändert.

Einsetzen auf Datei (Edit Menu) / **<ESC>Y**

schreibt den im Zwischenspeicher enthaltenen Text auf eine anzugebende Datei. Ist bereits eine Datei gleichen Namens vorhanden, wird diese ohne Warnung überschrieben.

Löschen (Edit Menu)

löscht den Bereich, *ohne* diesen in den Zwischenspeicher zu übertragen.

Suchen und Ersetzen eines Textes

Suchen (Befehle Menu) / **<CTRL-S>** / **<F3>**

sucht den angegebenen Text im Textspeicher. Ab der Cursorposition wird vorwärts gesucht, also in Richtung Textspeicherende. Ist der Text im Textspeicher vorhanden, springt der Cursor unmittelbar hinter den gefundenen Text, anderenfalls erscheint die Meldung "`Nicht gefunden`" (wie immer auf der Statuszeile).

Suchen rückwärts <CTRL-R>

sucht den angegebenen Text im Textspeicher. Die Suchrichtung ist rückwärts, d.h. von der Cursorposition an in Richtung Textspeicheranfang. Ist der Text im Textspeicher vorhanden, springt der Cursor auf das erste Zeichen des gefundenen Textes.

Ersetzen (Befehle Menu) / **<ESC>%** / **<SHIFT-F3>**

verlangt zunächst den Suchtext, und danach die Angabe, durch welchen Text dieser zu ersetzen ist. m2emacs sucht den angegebenen Text im Textspeicher, wiederum beginnend mit der Cursorposition. Wird der Text gefunden, springt der Cursor hinter den gefundenen Text, und es erscheint die Meldung "`ersetzen?`". Die Eingabe "`j`" oder " " ersetzt den Text und sucht das nächste Vorkommen des Ursprungtextes, "`n`" und <DEL> fahren ohne Ersetzung mit der Suche fort. Die Eingabe "`a`" lässt alle weiteren Vorkommen des Ursprungtextes ohne Bestätigung ersetzen. Sollen keine weiteren Ersetzungen mehr vorgenommen werden, kann mit <CTRL-G> abgebrochen werden.

Ändern eines Textbereichs

Mache Kleinbuchstaben <CTRL-X><CTRL-L>

ersetzt alle grossen Buchstaben des Bereichs (einschliesslich der Umlaute) durch den entsprechenden Kleinbuchstaben. Ziffern, Kleinbuchstaben und Sonderzeichen bleiben unverändert.

Mache Grossbuchstaben <CTRL-X><CTRL-U>

ersetzt alle kleinen Buchstaben des Bereichs (einschliesslich der Umlaute) durch den entsprechenden Grossbuchstaben. Ziffern, Grossbuchstaben und Sonderzeichen bleiben unverändert. Dieser Befehl ist nicht die Rücknahme des "Mache Kleinbuchstaben"-Befehls! Beide sind mit grösster Sorgfalt anzuwenden, da sie nur mühsam von Hand rückgängig gemacht werden können.

Mache Wort mit Kleinbuchstaben <ESC>L

ersetzt die Buchstaben des Wortes durch entsprechende Kleinbuchstaben und rückt den Cursor an das Ende des Worts. Gilt nur für den Rest des Worts, falls sich der Cursor mitten in einem Wort befindet.

Mache Wort mit Grossbuchstaben <ESC>U

ersetzt die Buchstaben des Wortes durch entsprechende Grossbuchstaben. Ansonsten gleichbedeutend mit dem vorherigen Befehl.

Mache Wort gross <ESC>C

ersetzt den Buchstaben unter dem Cursor durch den entsprechenden Grossbuchstaben und der Rest des Wortes wird in Kleinbuchstaben verwandelt. Danach befindet sich der Cursor hinter dem Wort. Stand der Cursor vor dem Aufruf nicht in einem Wort, wird die Operation auf das nachfolgende Wort ausgeführt.

Vertausche die letzten beiden Buchstaben <CTRL-T>

vertauscht den Buchstaben, der unmittelbar unter dem Cursor steht, mit dem unmittelbar davorstehenden. Dieser Befehl ist äusserst nützlich für den wohl häufigsten Fall von Tippfehler (Wer schreibt schon fehlerfrie?). Dieser Befehl funktioniert nicht am Zeilenende sowie am Zeilenbeginn, das Zeilenvorschubzeichen kann nicht vertauscht werden.

Schalterstellungen

Schalterstellungen beeinflussen die Funktionsweise einiger Befehle. Schalter können nicht explizit an- oder abgestellt, sondern nur umgestellt werden. Die gegenwärtige Schalterstellung ist im Menu sichtbar. Ein gesetzter, aktivierter Schalter wird durch ein spezielles Symbol ("√"), Checkmark genannt, vor dem jeweiligen Menueintrag angezeigt. Das Verändern einer Schalterstellung hat keinen Einfluss auf einen bereits bestehenden Textspeicherinhalt.

Automatisches Einrücken (Optionen-Menu) / **<CTRL-X>I**

Dieser Schalter beeinflusst das Verhalten des Editors, wenn ein Zeilenumbruch eingefügt wird. Ist der Schalter gesetzt, beginnt die neue Zeile mit gleich vielen Leer- und Tabulatorzeichen, wie die über ihr liegende Zeile. So fällt ein grosser Teil des (sich auszahlenden) Einrückaufwands weg. Ist der Schalter nicht gesetzt, beginnt jede neue Zeile in der ersten Spalte. Der Schalter wird beim Aufstarten des m2emacs gesetzt.

Gross/Klein (Optionen-Menu) / **<CTRL-X>C**

Dieser Schalter ist nur bei Suchvorgängen oder Ersetzungen von Bedeutung. Ist dieser Schalter gesetzt, wird zwischen Gross- und Kleinbuchstaben unterschieden. Diese Unterscheidung gilt auch für alle Umlaute. Dieser Unterscheidungsschalter ist beim Programmstart gesetzt.

Erzeuge Ikone (Optionen-Menu) / **<CTRL-X>W**

Dieser Schalter gibt an, ob beim Sichern einer Datei auch eine Ikone erzeugt werden soll oder nicht. Der gesetzte Schalter bewirkt ein Erzeugen einer Ikone allerdings nur dann, wenn nicht schon eine Ikone existiert. Benutzereigene Ikonen werden so *nicht* überschrieben. Dieser Schalter ist gesetzt, falls m2emacs von der Workbench aus gestartet wird.

Behalte Backup (Optionen Menu)

Ist dieser Schalter gesetzt, legt m2emacs beim Zurück- und Überschreiben einer Datei eine Sicherheitskopie an. Diese Sicherheitskopie hat den Namen der überschriebenen Datei, erweitert um den Buchstaben "O". Stolze Besitzer einer Harddisk können diesen Schalter setzen, um auch noch die (zuweilen wertvolle) zweitletzte Version greifbar zu haben. Für Diskettenbenutzer ist der Platzbedarf für Backups meist nicht vorhanden, weshalb den Schalter ausschalten sollten.

Bildschirminhalt auswählen und Fensterbehandlung

Erneuere Bildschirminhalt <CTRL-L>

schreibt die ganze Seite neu.

Verschiebe Text nach oben <CTRL-X><CTRL-N> / <F9>

schiebt den Text im Fenster um eine Zeile nach oben, d.h. die oberste Zeile verschwindet und eine neue Zeile wird sichtbar. Ist der Cursor vor der Bewegung auf der obersten Bildschirmzeile, wird er nachher wieder frisch zentriert.

Verschiebe Text nach unten <CTRL-X><CTRL-P> / <SHIFT-F9>

schiebt den Text im Fenster um eine Zeile nach unten.

Setze aktuelle Zeile an Fensteranfang <ESC>H / <ESC>!

verschiebt den Text, so dass sich die aktuelle Zeile am oberen Fensterrand befindet. Dabei bleibt die Cursorposition innerhalb der Zeile unverändert.

Teile Fenster (Befehle-Menu) / **<CTRL-X>2 / <F7>**

teilt das aktuelle Fenster in zwei. Beide Fenster haben den gleichen Textspeicherinhalt, können aber unabhängig voneinander einen neuen Textspeicher einlesen. Die Anzahl der Fenster ist nur durch den Bildschirm begrenzt.

Schliesse Fenster (Befehle-Menu) / **<CTRL-X>0 / <SHIFT-F7>**

schliesst das aktuelle Fenster und lässt das obige Fenster – im Falle des obersten das untere – entsprechend grösser werden. Das Schliessen eines Fensters hat keine direkte Einwirkung auf den vormals angezeigten Textspeicher. Dieser ist immer noch vorhanden und auch abrufbar.

Nur ein Fenster (Befehle-Menu) / **<CTRL-X>1**

lässt das aktuelle Fenster über den ganzen Bildschirm gehen.

Vergrössere Fenster <CTRL-X>Z / <SHIFT-F8>

vergrössert das aktuelle Fenster um eine Zeile, d.h. die Begrenzungszeile des Fensters verschiebt sich um eine Zeile nach unten. Wird das unterste Fenster vergrössert, wandert die Begrenzungszeile des darüberliegenden Fensters um eine Zeile nach oben.

Verkleinere Fenster <CTRL-X><CTRL-Z> / <F8>

verkleinert das aktuelle Fenster um eine Zeile. Ein Fenster kann bis auf eine Zeile reduziert werden.

Makrobefehle

Makros erlauben, immer wiederkehrende *Tastaturfolgen* auf einen Tastendruck abrufbar zu machen. Es ist immer nur eine Makrodefinition möglich; die Definition kann jedoch beliebig oft erneuert werden.

Makros können nicht verschachtelt werden. Die maximale Anzahl von abspeicherbaren Tastendrucken ist 256.

Beginne Makro (Befehle-Menu) **<CTRL-X>(/ <F4>**

zeigt an, dass eine neue Makrodefinition eingegeben wird. Nebst der Meldung auf der Statuszeile ändert auch der Bildschirmtitel zu "`m2emacs Makro Definition`". Nun kann eine beliebige Tastaturfolge eingeben werden ("Beginne Makro" und "Führe Makro aus" sind tabu). Alle Aktionen werden sofort ausgeführt und gleichzeitig auch intern vermerkt. Ist die gewünschte Ablauffolge beendet, muss der Befehl "Beende Makro" ausgeführt werden.

Beende Makro (Befehle-Menu) **<CTRL-X>) / <SHIFT-F4>**

schliesst die Makrodefinition ab. Die eingegebene Tastenfolge ist nun mit einem einzigen Tastendruck ausführbar.

Makro ausführen (Befehle-Menu) **<CTRL-X>E / <F5>**

führt die vorher definierte Tastenfolge aus. Vorsicht ist allerdings bei allen Befehlen geboten, die sich je nach Charakteristik der Zeile verschieden verhalten, beispielsweise der Befehl "Lösche bis Ende Zeile" bei leeren Zeilen. Versuchen Sie, solche Befehle in Makrodefinitionen zu vermeiden.

Wie leistungsfähig die Makros sind, wird im nachfolgen Abschnitt "Beispiel" gezeigt.

Modula-2 Befehle

Beim Öffnen einer Datei (also möglicherweise beim Aufstarten des m2emacs) wird ebenfalls nach einer Fehlerdatei gesucht. Eine Fehlerdatei eines Modula-2 Programms wird durch den Compiler erzeugt. Der Name der Fehlerdatei ergibt sich aus dem Dateinamen, erweitert um den Buchstaben "E". Wird eine solche Datei gefunden, liest m2emacs ebenfalls die Datei "Modula-2 Fehlermeldungen" im "s:"-

Verzeichnis ein. Da der Compiler nur Fehlernummern ablegt, benötigt man diese Datei, um die Fehlernummern in eine Klartextmeldung zu übersetzen.

Nächster Fehler (Modula-2 Menu) / **<F2>**

zeigt von der Cursorposition an das nächste Vorkommen eines Fehlers. Befindet sich zwischen der Cursorposition und dem Textspeicherende kein Fehler mehr, wird die Fehlersuche beim Textspeicheranfang fortgesetzt. Nach dem Anzeigen des Fehlers wird dieser aus der internen Fehlerliste gestrichen, d.h. jeder Fehler wird nur einmal angezeigt. Nachdem alle Fehler "gestrichen" sind, erscheint die Meldung "`Kein (weiterer) Fehler gefunden`".

Lies Fehlerliste (Modula-2 Menu) / **<SHIFT-F2>**

versucht die Fehlerdatei zum aktuellen Textspeicher erneut zu lesen. Dies ist nützlich, um schon "gestrichene" Fehler nochmals anzeigen zu können. Andererseits kann so auch eine neu erzeugte Fehlerdatei eingelesen werden, die der Compiler im Hintergrund frisch erzeugt haben könnte.

Verschiedene Befehle

Zeige Liste aller Textspeicher <CTRL-X><CTRL-B>

zeigt in einem eigenem Fenster eine Liste aller Textspeicher. Der Befehl gibt Auskunft über die Textspeichergrösse, Textnamen und den zugehörigen Dateinamen. In der ersten Spalte wird mit einem Stern (*) angezeigt, ob der Textspeicher verändert wurde.

Vertausche Cursor mit Marke <CTRL-X><CTRL-X>

setzt den Cursor an die Stelle der Marke bei gleichzeitigem Verschieben der Marke an die alte Cursorposition. Die Marke kann so als Merkpunkt eingesetzt werden, zu dem man jederzeit wieder schnell zurückkehren kann.

Abbruch einer Eingabe <CTRL-G> / <SHIFT-F5>

bricht die aktuelle Eingabe ab. Wurde irrtümlich ein Befehl aufgerufen, den man lieber nicht aufgerufen hätte, so kann die Eingabe durch <CTRL-G> abgebrochen werden, und m2emacs kehrt wieder in den Normalmodus zurück, ohne den unerwünschten Befehl auszuführen.

Info <CTRL-X>= / <SHIFT-F6>

zeigt einige Informationen zum aktuellen Textspeicher sowie zum Bildschirminhalt an. Die einzelnen Positionen haben dabei folgende Bedeutung:

Linien:	Die Anzahl der Zeilen des Textspeichers.
Linie:	Die Zeilennummer, auf der sich der Cursor befindet. Für diesen Wert gilt: $0 \leq \text{Linie} \leq \text{Linien}$.
Zeile:	Die Bildschirmzeile, auf der sich der Cursor befindet. Die oberste Zeile ist die Zeile 0.
Spalte:	Die Bildschirmspalte, auf der sich der Cursor befindet. Die Spaltenzählung beginnt ebenfalls mit 0.
CH:	Der Ordinalwert des unter dem Cursor liegenden Zeichen in Hexadezimaldarstellung.
.:	Die Nummer des Zeichens, auf dem der Cursor steht. Das erste Zeichen des Textspeichers hat die Nummer 0.

Die letzte Angabe zeigt die relative Cursorposition im Verhältnis zum ganzen Textspeicher, dessen Zeichenzahl auch angezeigt wird.

Setze Zähler <CTRL-U> / <HELP>

setzt einen Zähler für den nächsten Befehl. Auf der Statuszeile erscheint der Text "`Zähler: 4`". Wird "Setze Zähler" erneut aufgerufen, wird der aktuelle Zählerwert mit vier multipliziert. Stattdessen kann auch eine Zahl eingegeben werden.

Einige Beispiele (Nach dem Drücken von <CTRL-U>):

`Zähler: 12` ("1", "2" gedrückt)

`Zähler: 12` ("3", <CTRL-U> gedrückt)

m2emacs nimmt diesen Zähler für den nächsten Tastendruck. Um 12 Zeilen einzufügen, setzen Sie den Zähler auf 12 und drücken Sie <RETURN>. Um 78 Sterne in den Text einzufügen, können Sie, anstatt sie alle einzeln zu tippen, einfach den Zähler auf 78 setzen und die "*"-Taste drücken.

Übernimm Zeichen unverändert <CTRL-Q>

ermöglicht die Eingabe solcher Zeichen, die sonst nicht in den Text eingefügt werden können. Darunter fallen beispielsweise die Kontrollzeichen. Das Kommando erlaubt die Eingabe *eines* Sonderzeichens.

Betriebssystemumgebung

Wechsle Verzeichnis <CTRL-X>D(Befehle-Menu) / (Laufzeitoption -d)

setzt das Verzeichnis so, als wäre m2emacs in diesem Verzeichnis aufgerufen worden. Das Verzeichnis ist jedoch nur während der Laufzeit (und natürlich nur bis zum nächsten Wechsel) gültig und hat keinen Einfluss auf die Umgebung, von der m2emacs aufgerufen wurde. Dieser Befehl erleichtert die Eingabe von Dateinamen, wenn man viele Dateien aus demselben Verzeichnis bearbeiten möchte.

CLI-Befehl <CTRL-X>!

führt einen CLI-Befehl aus. So kann beispielsweise das Verzeichnis einer Diskette angeschaut werden, ohne m2emacs zu verlassen. Nebst dem gewünschten Kommando muss zusätzlich der "`run`"-Befehl entlang dem vorgegeben Pfad vorhanden sein.

Neues CLI (Befehle-Menu) / **<CTRL-C>**

startet ein neues CLI auf. Für die Rückkehr in m2emacs muss dieses CLI ganz normal mit "`endcli`" verlassen werden. Auch dieser Befehl benötigt den "`run`"-Befehl.

Optionen beim Aufruf

m2emacs kennt zwei Optionen, die beim Starten aufgerufen werden können.

-d Verzeichnis — wechselt das Verzeichnis, auf dem m2emacs arbeitet. Diese Option ist gleichbedeutend mit einem Aufruf von "Wechsle Verzeichnis" unmittelbar nach dem Aufstarten des Editors. Wird beim Starten auch ein Dateiname angegeben, wird die Datei sowie die zugehörige Fehlerdatei aus diesem Verzeichnis geladen.

-b — ist diese Option gesetzt, wird die alte Datei nicht als Sicherungskopie behalten.

-i — setzt den Schalter für die Ikonenerzeugung so, dass beim Abspeichern *keine* Ikone erzeugt wird. Dieser Schalter kann auch innerhalb des Editors wieder umgestellt werden.

Rückgabewerte

ok — Es wurden keine Dateien verändert.

warn Mindestens eine der Dateien wurde editiert.

Ein Beispiel

Um zu zeigen, wie Makros das Editieren erleichtern können, sei die Lösung eines besonders häufigen Problems dargestellt. Es geht darum, gemäss dem Wert einer Variablen, deren Typ ein Aufzählungstyp ist, einen entsprechenden Text auszugeben. Sei also

```
TYPE
  Farbe=(weiss,schwarz,rot,gruen,blau,gelb,lila,gold);
VAR
  farbe: Farbe;
```

dann erledigt folgende Tastaturfolge die Arbeit äusserst schnell. Als Vorbereitung kopieren Sie einfach obige zweite Zeile an den gewünschten Ort der Ausgabeanweisung und setzen "Automatisches Einrücken".

<CTRL-A><ALT-DEL>CASE farbe OF<DEL><DEL><RETURN>
<F4>| <ESC>.<ALT-→><ESC>W: WriteString(´<CTRL-Y>´);
<DEL><SHIFT-F4><F5><F5><F5><F5><F5><F5><F5>END;

Da das mit Sicherheit sehr kryptisch wirkt, sollte man das Beispiel sofort nachvollziehen.

Befehlszusammenfassung

Cursor-Bewegung

Buchstabe	<←>	<→>
Zeile	<↑>	<↓>
Wort	<ALT-←>	<ALT-→>
Seite	<ALT-↑>	<ALT-↓>
Anfang/Ende Zeile	<SHIFT-←>	<SHIFT-→>
Anfang/Ende Textspeicher	<SHIFT-↑>	<SHIFT-↓>

Lösch-Tasten

Buchstabe zurück/vor	<BACKSPACE>	<DEL>
Wort zurück/vor	<ALT-BACKSPACE>	<ALT-DEL>
Zeile zurück/vor	<SHIFT-BACKSPACE>	<SHIFT-DEL>

Funktionstasten

F1	Öffne neue Datei
<SHIFT-F1>	Sichere aktuellen Textspeicher
F2	Zeige nächster Fehler
<SHIFT-F2>	Lies Fehlerliste erneut ein
F3	Suche
<SHIFT-F3>	Ersetze
F4	Beginne mit Makro-Definition
<SHIFT-F4>	Schliesse Makro-Definition ab
F5	Makro ausführen
<SHIFT-F5>	Abbruch der aktuellen Eingabe
F6	Springe auf Zeile
<SHIFT-F6>	Information
F7	Teile Fenster
<SHIFT-F7>	Schliesse Fenster
F8	Verkleinere das aktuelle Fenster
<SHIFT-F8>	Vergrössere das aktuelle Fenster
F9	Verschiebe Text nach oben (vorwärts)
<SHIFT-F9>	Verschiebe Text nach unten (rückwärts)
F10	Lösche aktuelle Zeile
<SHIFT-F10>	Füge oberhalb neue Zeile ein

Control-Befehle

<CTRL-@>	Setze Marke
<CTRL-A>	Gehe zum Zeilenanfang
<CTRL-B>	Gehe einen Buchstaben zurück
<CTRL-C>	Starte CLI
<CTRL-D>	Lösche Buchstaben unter dem Cursor
<CTRL-E>	Gehe zum Zeilenende
<CTRL-F>	Gehe einen Buchstaben vorwärts
<CTRL-G>	Breche aktuelle Eingabe ab
<CTRL-H>	Lösche Buchstaben vor dem Cursor
<CTRL-I>	Tabulator
<CTRL-J>	Füge Zeilenumbruch ein
<CTRL-K>	Lösche bis zum Zeilenende, falls Zeile leer, lösche Zeile
<CTRL-L>	Erneuere Bildschirminhalt
<CTRL-M>	Füge Zeilenumbruch ein
<CTRL-N>	Gehe eine Zeile vorwärts
<CTRL-O>	Füge neue Zeile oberhalb ein
<CTRL-P>	Gehe eine Zeile rückwärts
<CTRL-Q>	Übernimm nächsten Buchstaben unverändert
<CTRL-R>	Suche rückwärts
<CTRL-S>	Suche vorwärts
<CTRL-T>	Vertausche die letzten beiden Buchstaben
<CTRL-U>	Setze Zähler
<CTRL-V>	Gehe eine Seite vorwärts
<CTRL-W>	Bereich ausschneiden
<CTRL-X>	erweiterte Befehle (→ unten)
<CTRL-Y>	Zwischenspeicher einsetzen
<CTRL-Z>	Sichere und Verlasse

Erweiterte Befehle

<CTRL-X><CTRL-B>	Zeige eine Liste aller Textspeicher
<CTRL-X><CTRL-C>	Verlasse den Editor
<CTRL-X><CTRL-D>	Lösche aktuelleZeile
<CTRL-X><CTRL-F>	Neuer Dateinamen
<CTRL-X><CTRL-J>	Füge Zeilen zusammen
<CTRL-X><CTRL-L>	Mache Kleinbuchstaben in Bereich
<CTRL-X><CTRL-M>	Zeige nächsten Fehler
<CTRL-X><CTRL-N>	Verschiebe Text nach oben (vorwärts)
<CTRL-X><CTRL-P>	Verschiebe Text nach unten (rückwärts)
<CTRL-X><CTRL-R>	Öffne Datei

<CTRL-X><CTRL-S>	Sichere aktuellen Textspeicher
<CTRL-X><CTRL-U>	Mache Grossbuchstaben in Bereich
<CTRL-X><CTRL-V>	Lies Datei
<CTRL-X><CTRL-W>	Sichere als neue Datei
<CTRL-X><CTRL-X>	Vertausche Cursor mit Marke
<CTRL-X><CTRL-Z>	Verkleinere das aktuelle Fenster
<CTRL-X>!	führe einen CLI-Befehl aus
<CTRL-X>=	Info
<CTRL-X>(	Beginne mit Makro-Definition
<CTRL-X>)	Schliesse Makro-Definition ab
<CTRL-X>0	Schliesse Fenster
<CTRL-X>1	Nur ein Fenster
<CTRL-X>2	Teile Fenster
<CTRL-X>B	Wechsle Textspeicher
<CTRL-X>C	Schalte Gross/Klein-Unterscheidung um
<CTRL-X>D	Wechsle Verzeichnis
<CTRL-X>E	Makro ausführen
<CTRL-X>I	Schalte automatisches Einrücken um
<CTRL-X>K	Lösche einen Textspeicher
<CTRL-X>N	Gehe zum nächsten Fenster
<CTRL-X>P	Gehe zum vorhergenden Fenster
<CTRL-X>W	Schalte Ikonenerzeugung um
<CTRL-X>Z	Vergrössere das aktuelle Fenster
<CTRL-X><DEL>	Lösche aktuelle Zeile

ESC-Befehle

<ESC><BACKSPACE>	Lösche ein Wort zurück
<ESC>!	Setze aktuelle Zeile an oberen Fensterrand
<ESC>.	Setze Marke
<ESC>>	Gehe zum Ende des Textspeichers
<ESC><	Gehe an den Anfang des Textspeichers
<ESC>B	Gehe ein Wort zurück
<ESC>C	Mache Wort gross
<ESC>D	Lösche ein Wort vorwärts
<ESC>F	Gehe ein Wort vorwärts
<ESC>H	Setze aktuelle Zeile an oberen Fensterrand
<ESC>L	Mache Wort mit Kleinbuchstaben
<ESC>M	Gehe zur Zeile
<ESC>U	Mache Wort mit Grossbuchstaben
<ESC>V	Blättere Seite zurück
<ESC>W	Bereich kopieren

<ESC>Y	Bereich auf Datei kopieren
<ESC>%	Ersetzen

5 Der Compiler

Aufruf 3

Arbeitsweise des Compilers 3

Optionen 5

Einschränkungen 9

Rückgabewerte 14

Aufruf

```
m2c [-dfinqrsv] {Dateiname}
```

Arbeitsweise des Compilers

Die Aufgabe eines Compilers ist die Übersetzung eines für den Menschen lesbaren Quelltextes in einen maschinennäheren Code gleicher Bedeutung. Diese recht einfache Erklärung beinhaltet aber nicht den Umfang dieser Aufgabe. Die Übersetzung eines Quelltextes muss in mehrere Teilaufgaben gegliedert werden. Zunächst müssen die Symbole erkannt, danach die Syntax kontrolliert, darauf die internen Tabellen für die verwendeten Strukturen generiert, die Typenkontrolle durchgeführt und schliesslich der Code erzeugt werden. Für ein Fehlerprotokoll ist unter Umständen noch ein weiterer Durchgang nötig.

Ein Mehrpasscompiler führt jeden Arbeitsgang einzeln aus. Dazu müssen Zwischendateien erzeugt und von jedem Durchgang wieder gelesen werden. Dies führt zu einer langsamen Compilation und, falls im Hauptspeicher zwischengespeichert wird, zu einem erhöhten Speicherbedarf.

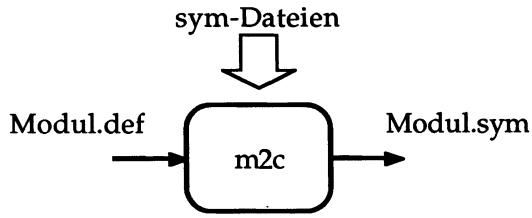
Ein Einpasscompiler führt diese Arbeitsgänge parallel aus. Jedes Programmelement, dessen Struktur erkannt ist, wird sofort in Code umgewandelt. Ein Einpasscompiler wird dadurch extrem schnell, ohne dass dem Programmierer Nachteile erwachsen. Einzig alle referenzierten Objekte müssen vor dem Gebrauch deklariert sein.

Der Amiga Modula-2 Compiler ist ein Einpass-Compiler. Der erzeugte Code ist MC68000-Maschinencode, was sehr schnelle Programme bewirkt.

Die Quelldateien sind normale Textdateien. Wird ein Definitionsmodul compiliert, erzeugt der Compiler eine Symbol-Datei. Aus Implementationsmoduln oder Programmoduln entstehen Objekt-Dateien.

Eine Symbol-Datei dient als Referenz bei einem Import dieses Moduls. Es handelt sich um eine übersetzte Version der Schnittstelle. Um eine vollständige Typen- und Versionskontrolle zu ermöglichen, ist ein eindeutiger Schlüssel abgelegt. Importiert ein Modul ein anderes Modul, so wird für die Compilation dessen Symboldatei gelesen. Dies ermöglicht eine vollständige Typenkontrolle.

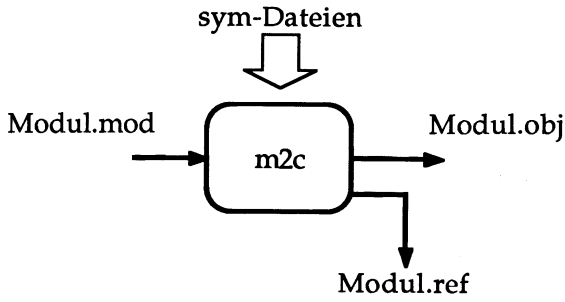
Umgebung für Definitions-Moduln



Objekt-Dateien aus Implementationsmoduln oder aus Programmoduln werden gleich behandelt. Bei Implementationsmoduln wird kontrolliert, ob alle in der Definition vermerkten Prozeduren auch implementiert sind. Ebenfalls wird die Gleichheit des Prozedurkopfs in Definition und Implementation geprüft. Die Schlüssel aller importierten Moduln werden vermerkt. So kann beim Linkvorgang kontrolliert werden, ob die Definition eines importierten Moduls nicht zwischenzeitlich geändert wurde.

Zu jeder Objekt-Datei wird auch eine sogenannte Referenz-Datei erzeugt. Die Referenz-Datei enthält Informationen, die vom Debugger benötigt werden. Für die Compilation und den Linkvorgang sind sie ohne Bedeutung.

Umgebung für Implementations- oder Programm-Moduln



Wird während der Compilation ein Fehler in der Quelldatei gefunden, bricht die Übersetzung *nicht* ab. Es wird versucht, mit vernünftigen Annahmen weiter zu übersetzen. So können mehrere Fehler des Quelltextes in *einem* Übersetzungsvorgang gefunden werden. Alle gefundenen Fehler werden in einer vom Compiler erzeugten Datei festgehalten. Diese Datei hat den Namen der Eingabedatei, erweitert um den Buchstaben "E". Wird beispielsweise das Modul "InOut.mod" übersetzt, so erzeugt der Compiler eine Datei "InOut.modE" im gleichen Verzeichnis. Diese Fehlerdatei wird dann vom Editor oder vom Fehlerlister eingelesen, um alle Fehler anzuzeigen.

Optionen

Optionen, im Deutschen vielleicht häufiger Schalter genannt, sind zusätzliche Anweisungen an den Compiler. Sie repräsentieren Boole'sche Werte. Ist der Wert wahr, werden spezielle Aktionen ausgeführt, ist er falsch, unterbleiben diese Aktionen.

Die Compileroptionen werden in die Kommentare des Quelltextes eingebettet. Sie können an jeder Stelle in einem nicht verschachtelten Kommentar auftreten. In einem geschachtelten Kommentar werden sie ignoriert. Kommentare dürfen beliebig viele Compileranweisungen (-optionen) enthalten.

Die Syntax einer Compileranweisung ist denkbar einfach. Ein $-Zeichen wird von einem Buchstaben und einem der drei Zeichen "+", "-" oder "="

gefolgt. Wird diese Syntax nicht eingehalten (ein "_" statt einem "-" oder Leerzeichen zwischen diesen Zeichen), oder existiert keine Compileranweisung mit dem vorgegeben Buchstaben, so wird keine Aktion ausgeführt. Wird eine bereichsbezogene Option falsch angewendet (Einschalten, Zurücksetzen oder falscher Ort), wird eine Fehlermeldung erzeugt.

Beispiele für Compileroptionen:

```
MODULE CompOpt;
(* Achtung: $R+ (* gültige Option *) *)
(* keine Option: (* $V- *)
  ( verschachtelter Kommentar ) *)
(* wird ignoriert: $S + (Leerzeichen zuviel) *)
CONST
  text = "$R-"
  (* keine Option, da nicht in Kommentar *)
BEGIN
...
```

Es gibt zwei Arten von Compileroptionen: stapelbare und bereichsbezogene. Stapelbare Optionen können immer wieder ein- und ausgeschaltet werden, ausserdem ist der Zustand vor der Veränderung vermerkt. Sie dürfen an beliebiger Stelle des Quelltextes stehen. Bereichsbezogene Compileroptionen sind nur für einen Gültigkeitsbereich, d.h im Normalfall eine Prozedur, gültig. Sie können nicht beliebig umgestellt, sondern nur für eine Prozedur aktiviert werden, und befinden sich nach der Compilation der Prozedur automatisch wieder im Ausgangszustand.

Alle Optionen sind beim Starten des Compilers gesetzt. Dadurch wird ein sicherer Ablauf des erzeugten Programms garantiert. Stapelbare Optionen können beim Aufruf des Compilers vom CLI aus in der gewohnten Weise ausgeschaltet werden. Zusätzlich kann auch die -d Option beim Aufruf gewählt werden.

Stapelbare Optionen können durch folgende Anweisung ein- beziehungsweise ausgeschaltet werden:

(* ... $R+ (hier wird eingeschaltet) ... $R- (und hier ausgeschaltet) ...*)

Dabei wird der alte Wert vermerkt. Er wird durch folgende Compileranweisung wieder aktiv:

```
(* ... $R= ... *)
```

Für bereichsbezogene Optionen ist nur das Ausschalten möglich:

```
(* ... $I- ... *)
```

Als Bereich kann entweder das ganze Modul oder eine Prozedur gelten. Jede bereichbezogene Option gibt jedoch nur für eins von beiden einen Sinn.

Bereichskontrolle (R; stapelbar)

Wird auf einen Teilbereich ("subrange" wie [0..31] oder ein Feld wie a[3]) zugegriffen, so generiert der Compiler Code um die Gültigkeit zu überprüfen. Unter- oder Überschreiten des Bereichs unterbricht das Programm mit einem Laufzeitfehler. Ist das Programm vorsichtig ausgetestet (und nur dann), kann die Kontrolle mit `(*$R-*)` ausgeschaltet werden, um etwas Geschwindigkeit zu gewinnen.

```
VAR
  i: INTEGER;
  zwerg: ARRAY [1..7] OF INTEGER;
FOR i:= 1 TO 7 DO
(* $R- *) zwerg[i]:=0 (* $R= *)
END;
```

Die Bereichskontrolle wird hier unmittelbar vor der Zuweisung der Feldvariablen abgeschaltet und anschliessend wieder auf den alten Wert zurückgesetzt. Dabei muss der alte Wert nicht bekannt sein, da dieser vom Compiler zwischengespeichert ("auf den Stapel gelegt") wurde. Das Gleichheitszeichen (=) holt diesen vorgängigen Wert.

Über- und Unterlaufskontrolle (V; stapelbar)

Die interne Zahlenrepräsentation kennt keine zu grossen oder zu kleinen Zahlen. Jede Über- oder Unterschreitung lässt sich jedoch durch das Testen eines speziellen Prozessorregisters (→ [MC68000], condition code) feststellen. Der Compiler generiert diesen Code, der im Über- oder Unterlaufsfall das Programm mit einem Laufzeitfehler unterbricht. Ist das Programm vorsichtig ausgetestet, kann diese Kontrolle mit `(*$V-*)` ausgeschaltet werden.

Stacküberlaufskontrolle (S; stapelbar)

Das Betriebssystem kontrolliert nicht, ob ein Prozess den ihm zugewiesenen Speicherplatz überschreitet. Ist dies der Fall, stellt sich meist ein unerklärlicher Laufzeitfehler ein, dessen Ursache kaum feststellbar ist. Amiga Modula-2 generiert Code zur Prüfung des Stackbereichs. Diese Prüfung muss bei jedem Prozeduraufruf durchgeführt werden, was den Code vergrössert. Werden keine Prozeduren rekursiv aufgerufen, oder ist sonst ein Stacküberlauf unwahrscheinlich, so kann die Stacküberlaufskontrolle mit `(*$S-*)` ausgeschaltet werden.

Bezeichnerkontrolle (N; stapelbar)

In einer Symbol-Datei werden alle Bezeichner der Prozedurparameter abgelegt und deren Übereinstimmung in der Implementation überprüft. Sollen die Bezeichner nicht vermerkt werden, kann in einem Definitionsmodul das Ablegen der Bezeichner mit `(*$N-*)` unterbunden werden. Sollen nur die abgelegten Bezeichner nicht überprüft werden, muss die Option `(*$N-*)` im Implementationsmodul gewählt werden.

Funktionsrückgabekontrolle (F; stapelbar)

Am Ende jeder Funktionsprozedur generiert der Compiler Code, der im Falle einer fehlenden RETURN-Anweisung ausgeführt wird. Dieser Code generiert einen Laufzeitfehler, der das Programm abbricht. Ist bei einer Funktionsprozedur die ordnungsgemässe Beendigung garantiert (und nur dann), kann mit `(*$F-*)` diese Kontrolle abgeschaltet werden.

Implementation vorhanden (M; bereichsbezogen pro Modul)

Zu jeder Symbol-Datei erwartet der Linker auch eine Objekt-Datei. Besteht jedoch die Definition nur aus Typen- und Konstantenvereinbarungen und ist kein Code beim Import dieses Moduls auszuführen (vom Compiler erst bei der *Implementation* feststellbar!), erübrigt sich eine Objekt-Datei. In einem solchen Fall ist *vor* dem Modulkopf des Definitionsmoduls die Compileranweisung `(*$M-*)` zu schreiben.

Parameter Deallokation (P; bereichsbezogen pro Prozedur)

In Modula-2 dealloziert jede Prozedur (d.h. baut den Stack wieder ab) ihre Parameter, da sie aus der Vereinbarung genau weiss, mit wievielen Parametern sie aufgerufen wurde. In C werden die Parameter vom Aufrufer dealloziert, da nur er weiss, wieviele Parameter er übergeben hat. Einige Betriebssystemprozeduren benötigen eine Prozedur als Parameter. Diese dürfen allerdings ihre Parameter *nicht* deallozieren, was mit `(*$P-*)` ausgeschaltet werden kann.

Entry/Exit-Code (E; bereichsbezogen pro Prozedur)

Jede Prozedur führt zu Beginn einen speziellen Code aus, der den Modulzeiger holt, für die lokalen Variablen Platz alloziert und den dynamischen Link erstellt. Am Ende der Prozedur wird der Stack wieder entsprechend abgebaut und der Modulzeiger restauriert. Diese beiden Codeteile werden durch `(*$E-*)` nicht erzeugt. Dies wird vor allem bei Assembler-Prozeduren (INLINE-Anweisungen) zur Optimierung benötigt. Eine Prozedur ohne Entry/exit-Code besitzt auch keine RTS-Anweisung am Ende der Prozedur.

Debug-Information, Referenz-Datei (-d; Befehlszeile)

Der Compiler erzeugt zu jeder Objekt-Datei auch eine Referenz-Datei, die vom Debugger benötigt wird. Soll diese Datei nicht erzeugt werden, ist diese Option auf der Befehlszeile anzugeben.

Erzeuge Ikone (-i; Befehlszeile)

Soll der Compiler keine Ikone für die Objekt-Datei erzeugen, ist diese Option auf der Befehlszeile anzugeben.

Einschränkungen

Ein Compiler ist immer auch einigen Einschränkungen unterworfen, sei es durch die Implementation, das darunterliegende Betriebssystem oder durch die Hardware, auf der der Compiler läuft. Da das unvorbereitete Auftreten solcher Einschränkungen sehr ärgerlich sein kann, sind die Einschränkungen von Amiga Modula-2 im folgenden aufgelistet, auch wenn Sie kaum an die Grenzen stossen werden.

- Maximale Grösse von Code- und Konstantenbereich pro Modul: 32kByte.

  Der Code und die konstanten Zeichenketten müssen innerhalb eines Moduls im Bereich von 32k liegen. Die Gesamtgrösse eines Programms, d.h. eines komplett gelinkten Moduls, ist allerdings *nicht* eingeschränkt

- Bezeichnerbuffer pro Modul: 32kByte

  Alle Bezeichner, die entweder deklariert oder importiert werden, müssen vom Compiler abgelegt werden. Diese Ablage heisst Bezeichnerbuffer.

- Maximale Anzahl importierter Moduln pro Compilationseinheit: 64.

  Ein Programm darf aus beliebig vielen Moduln bestehen, vorausgesetzt jede Compilationseinheit importiert im Maximum 64 Moduln.

- Maximale Anzahl von Strukturen pro Definitionsmodul: 1024.

  Jeder neue Typ (Aufzählungstypen, Verbunde, Felder, Unterbereiche) bildet eine neue Struktur. Es spielt dabei keine Rolle, wie oft eine Struktur auftritt. Entscheidend ist einzig die Deklaration. Das strikte Einführen von Typnamen reduziert die Anzahl der Strukturen.

- Maximaler Speicherbedarf aller Variablen pro Gültigkeitsbereich: 2 GByte.

  Die Einschränkung pro Gültigkeitsbereich bedeutet, dass in einem Modul mit einem Variablenbedarf von 2 GByte auch jede Prozedur wiederum Variablen bis zur Grösse von 2 GByte deklarieren kann, wobei dann eine entsprechende Stack-Grösse gesetzt werden muss.

- Maximale Anzahl von Konstanten in Aufzählungstyp: 4 294 967 295.

- Maximale Länge des Modulnamens: 30 Zeichen.

- Maximale Anzahl EXIT-Anweisungen in LOOP-Anweisung: 16.

  Der Compiler kann bis zu 16 unbearbeitete EXIT-Anweisungen handhaben. Eine EXIT-Anweisung gilt dann als unbearbeitet, wenn die zur LOOP-Anweisung zugehörige END-Anweisung noch nicht erreicht wurde.

- Maximale Anzahl verschachtelter WITH-Anweisungen: 6.

- Ausdrucks-Verschachtelungstiefe: 5 REAL-Terme oder 2 LONGREAL-Terme.

- Maximale Anzahl von Fällen in einer CASE-Anweisung: 128.

  Oftmals kann die Anzahl von Fallunterscheidungen reduziert werden. Die Marke "`1..5:`" entspricht nämlich einem Fall, während die Marke "`1,2,3:`" drei Fällen entspricht.

- Maximale SET-Grösse: 32 Elemente.

  Gemäss der Modula-2 Definition ist die SET-Grösse durch die Wort-Grösse des Prozessors bestimmt. Obwohl das MC68000-Wort nur 16 Bit gross ist (BITSET), kann er auch 32-Bit "Long Words" verarbeiten (→ [MC68000]), was hier ausgenutzt wird.

- Funktionsresultate: ARRAY und RECORD Typen sind nicht erlaubt.

Zusammenfassung

Standardtypen

BOOLEAN	[FALSE, TRUE]
CARDINAL	$0 \leq card \leq 65535 = 2^{16}-1$
CHAR	0C ≤ char ≤ 377C
INTEGER	$-32768 = -2^{15} \leq int \leq 32767 = 2^{15}-1$
LONGCARD	$0 \leq longCard \leq 4294967295 = 2^{32}-1$
LONGINT	$-2147483648 = -2^{31} \leq longInt \leq 2147483647 = 2^{31}-1$
LONGREAL	$-1.0 \cdot 10^{308} \leq longReal \leq 1.0 \cdot 10^{308}$, Genauigkeit 10^{-15}
REAL	$-1.0 \cdot 10^{38} \leq real \leq 1.0 \cdot 10^{38}$, Genauigkeit 10^{-6}

Standardprozeduren

DEC(x)	vermindert x um 1 (nur skalare Typen)
DEC(x,n)	vermindert x um n (nur skalare Typen)
EXCL(s,i)	entfernt Element i aus dem Set s
HALT	bricht Programm ab
INC(x)	erhöht x um 1 (nur skalare Typen)
INC(x,n)	erhöht x um n (nur skalare Typen)
INCL(s,i)	fügt Element i in Set s ein

Standardfunktionen

ABS(x)	Absolutwert
CAP(c)	bildet Intervall "a.."z" auf "A.."Z" ab, sonst keine Änderung
CHR(x)	entsprechender Zeichenwert
FLOAT(x)	wandelt INTEGER-Wert in REAL-Wert um
HIGH(a)	obere Feldgrenze (nur bei "offenen Feldparametern")
MAX(T)	Grösstmöglichster Wert des Typs T
MIN(T)	Kleinstmöglicher Wert des Typs T
ODD(x)	gibt TRUE zurück, falls x ungerade
ORD(x)	Ordinalwert von x
SIZE(T)	Grösse (in Byte) des Typs T
SIZE(x)	Grösse der Variablen x
TRUNC(x)	gibt ganzahligen Teil von x zurück
VAL(T,x)	gibt Wert des Typs T mit Ordinalwert x zurück

Operatoren und Begrenzer

+	&	[	#	<>	}
-	.	{	<	..	|
*	,	^	>	:	
/	;	~	<=	)	
:=	(	=	>=	]	

Reservierte Worte

AND	EXPORT	QUALIFIED
ARRAY	FORWARD	RECORD
BEGIN	FROM	REM
BPOINTER	IF	REPEAT
BY	IMPLEMENTATION	RETURN
CASE	IMPORT	SET
CODE	IN	THEN
CONST	LOOP	TO
DEFINITION	MOD	TYPE
DIV	MODULE	UNTIL
DO	NOT	VAR
ELSE	OF	WHILE
ELSIF	OR	WITH
END	POINTER	
EXIT	PROCEDURE	

Vordefinierte Bezeichner

ABS	HALT	NIL
BOOLEAN	HIGH	ODD
CAP	INC	ORD
CARDINAL	INCL	PROC
CHAR	INTEGER	REAL
CHR	LONGCARD	SIZE
DEC	LONGINT	TRUE
EXCL	LONGREAL	TRUNC
FALSE	MAX	
FLOAT	MIN	

Rückgabewerte

ok	Der Befehl konnte ordnungsgemäss ausgeführt werden.
error	Mindestens einer der Dateien konnte nicht fehlerfrei übersetzt werden.

6 Der Linker

Aufruf 3

Arbeitsweise des Linkers 3

Optionen 3

Fehlermeldungen 4

Rückgabewerte 5

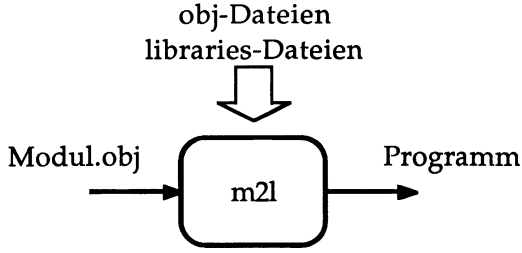
Aufruf

```
m2l [-iqx] {Dateiname}
```

Arbeitsweise des Linkers

Der Linker bindet Objektdateien zu einem ausführbaren Programm. Als Hauptmodul wird dasjenige Modul verwendet, dessen Objektdatei als Argument übergeben wird. Die Objektdateien aller referenzierten (importierten) Moduln werden dem Pfad folgend gesucht und eingelesen. Sind alle Referenzen aufgelöst, wird eine neue Datei geschrieben. Diese Datei enthält ein ausführbares Programm und besitzt den Namen des Hauptmoduls (ohne Erweiterung). Während des Linkvorgangs werden die Namen der gelesenen Dateien sowie der benötigten Amiga-Libraries ausgegeben.

Umgebung für den Linker



Optionen

-i Diese Option weist den Linker an, *keine* Ikone zu erzeugen.

-q Es werden *keine* Namen von gelesenen Dateien ausgegeben. Fehlermeldungen erscheinen wie normal.

-x Wird diese Option gewählt, erzeugt der Linker *keine* ausführliche Tabelle der gelinkten Moduln. Nur die Bibliotheken die vom Laufzeitsystem geöffnet werden müssen sind eingetragen.

Fehlermeldungen

Die Fehlermeldungen des Linkers werden am besten an einem Beispiel erklärt. Dabei wird folgender Modul-Anfang angenommen:

```
MODULE prog;
IMPORT ASCII, InOut;
```

Der Programmaufruf sei folgender:

```
m2l obj/prog.obj
```

Unverträglicher Schlüssel für 'ASCII' in 'InOut' aus 'obj/InOut.obj'

Da das Modul 'InOut' das Modul 'ASCII' ebenfalls importiert, ist in der Objekt-Datei des InOut-Moduls der Schlüssel des Moduls 'ASCII' vermerkt. Dieser Schlüssel wird bei der Compilation erzeugt, damit eine erneute Compilation eines Definitionsmoduls festgestellt werden kann. Erscheint die genannte Fehlermeldung, unterscheidet sich der ASCII-Schlüsseleintrag des Moduls 'prog' vom ASCII-Schlüsseleintrag des Moduls 'InOut'. Daraus folgt, dass das Definitionsmodul 'ASCII' zwischen den Compilationen von 'prog' und 'InOut' selbst compiliert wurde. Da die Gleichheit beider Versionen nicht garantiert werden kann, muss der Linkvorgang abgebrochen werden.

Verschiedene Modultypen für 'Dos' aus 'prog.obj'

Der Modultyp (normal, ohne Implementation, Library) muss vor dem Modulschlüssel geprüft werden. Werden zwei Moduln gleichen Namens aber verschiedenen Typs importiert, erscheint diese Meldung.

Speichermangel für 'Terminal'

Die Objekt-Datei des Moduls 'Terminal' kann nicht mehr in den Hauptspeicher geladen werden.

Fehlendes Modul 'Terminal'

Die Objekt-Datei des Moduls 'Terminal' ist in keinem Projekt entlang des Pfads vorhanden. Entweder ist der Pfad unvollständig, oder die Objekt-Datei ging verloren.

Formatfehler in 'obj/Terminal.obj'

Entweder es handelt sich bei dieser Datei nicht um eine Objekt-Datei, oder die Datei wurde durch einen Diskettenfehler unlesbar.

Rückgabewerte

ok	Der Befehl konnte ordnungsgemäss ausgeführt werden.
error	Mindestens bei einer Datei ist beim Binden ein Fehler aufgetreten.

7 Der Fehlerlister

Aufruf 3

Aufgabe und Arbeitsweise des Fehlerlisters 3

Optionen 3

Beispiel-Ausgaben 4

Fehlermeldungen 6

Rückgabewerte 6

Aufruf

```
m2error [-x] {Dateiname}
```

Aufgabe und Arbeitsweise des Fehlerlisters

Dieser Teil beschreibt die Möglichkeit, lesbare Fehlerlisten zu generieren. Dies ist nur nötig, wenn man statt m2emacs einen anderen Editor verwenden will.

m2error zeigt für jeden Fehler die unmittelbare Umgebung der fehlerhaften Textstelle an. Dazu werden etwa die letzten 100 Zeichen vor der Fehlerstelle auf den Bildschirm geschrieben. Ist davon bereits ein Teil auf dem Bildschirm vorhanden, so wird nur der neue Text geschrieben. In der nachfolgenden Zeile einer fehlerhaften Programmtextzeile zeigt ein "^"-Zeichen die Fehlerstelle an und auf den folgenden Zeilen die Fehlermeldungen im Klartext. Die Fehlerumgebung sollte ausreichen, den Ort genau zu lokalisieren und die Fehlerstelle zu finden. Um das Auffinden der Fehlerstellen noch zu vereinfachen, wird auch die Zeilennummer jeder Zeile angegeben.

m2error benötigt die Datei "`Fehler-Meldungen`" im "`M2:`"-Verzeichnis. Ist diese Datei nicht vorhanden, bricht das Programm ab.

Optionen

m2error kennt eine Option:

`-x` schreibt nicht nur den Programmtext an einer Fehlerstelle, sondern erstellt das gesamte Abbild des Ursprungtextes. Die Fehler werden in derselben Weise wie oben beschrieben dargestellt.

Beispiel-Ausgaben

Nachfolgend die Ausgabe von m2error, erstes Beispiel ist aufgerufen ohne die Option "-x", die zweite mit.

```
m2error, 3.2d, 1.11.88, © AMSoft

         1   MODULE err;
         2
         3   FROM InOut IMPORT
         4    WriteChar,WriteLn,WriteInt;
             |^
             |   55: Bezeichner nicht sichtbar

 ...

        15     a,b,c: INTEGER
        16    END;
        17
        18   BEGIN
        19    FOR i:=0 TO 99 DO
        20     FOR j:=0 TO 99 DO
        21      WriteInt(i, 6) WriteInt(j,6);
     WriteLn;
             |-----------------^
             |   96: Strichpunkt zwischen
     Anweisungen
                   erwartet

        22     END
        23    END;
        24    i:=p.a;
             |-----^
             |   43: Bezeichner sollte einen Record
                   bezeichnen, nicht POINTER-Typ
             |------^
             | 3028: Inkompatible Zuweisung
     Unterbereich
```

```
m2error, 3.2d, 1.11.88, © AMSoft

     1   MODULE err;
     2
     3   FROM InOut IMPORT
     4    WriteChar,WriteLn,WriteInt;
         |^
         |   55: Bezeichner nicht sichtbar

     5
     6   (*
     7    * Demo Programm für den Fehlerlister.
     8    *
     9    * Es enthält drei Fehler, die die
verschiedenen
    10    * Meldungsarten des Compiler
aufzeigen.
    11    *)
    12   VAR
    13    i,j: INTEGER;
    14    p: POINTER TO RECORD
    15     a,b,c: INTEGER
    16    END;
    17
    18   BEGIN
    19    FOR i:=0 TO 99 DO
    20     FOR j:=0 TO 99 DO
    21      WriteInt(i, 6) WriteInt(j,6);
WriteLn;
         |-----------------^
         |   96: Strichpunkt zwischen
Anweisungen
                erwartet

    22     END
    23    END;
    24    i:=p.a;
         |-----^
         |   43: Bezeichner sollte einen Record
               bezeichnen, nicht POINTER-Typ
         |------^
         | 3028: Inkompatible Zuweisung
Unterbereich
    25   END err.
    26
```

Fehlermeldungen

Modulname: Datei nicht gefunden

Die Eingabedatei wurde nicht gefunden, entweder existiert sie nicht oder der Name wurde falsch geschrieben.

Modulname: Keine Fehlerdatei gefunden

Zur angegebenen Quelldatei existiert keine Fehlerdatei. Auch mit der -x-Option wird dann *keine* Ausgabe erzeugt.

M2:Fehlermeldungen nicht gefunden oder zuwenig Speicher

Die Datei "`M2:Fehler-Meldungen`" konnte nicht geladen werden; entweder sie wurde nicht gefunden oder der benötigte Speicherplatz konnte nicht alloziert werden.

Rückgabewerte

ok	Der Befehl konnte ordnungsgemäss ausgeführt werden.
warn	Eine Datei oder deren Fehlerdatei wurde nicht gefunden.
fail	Die Datei "`M2:Fehler-Meldungen`" kann nicht geöffnet werden.

8 Anmerkungen zu Amiga Modula-2

Eigenschaften 3

Ganze Zahlen 3

Gleitpunkt-Zahlen 5

Restriktionen des Einpass-Compilers 6

Anordnung (Alignment) 6

Sonstiges 8

Systemspezifische Erweiterungen / SYSTEM 9

Eigenschaften

Dieses Kapitel beschreibt einige Eigenschaften von Amiga Modula-2. Das beinhaltet auch Unterschiede und Besonderheiten zu den üblichen Implementationen. Ebenfalls sollen hier allfällige Differenzen zu gängigen Lehrbüchern hervorgehoben werden. Die Unterschiede sind auf die ersten Standardisierungsschritte zurückzuführen[1].

Ganze Zahlen

Da der MC68000 Prozessor eine Registerbreite von 32 Bit hat, werden auch Rechengrössen von dieser Grösse angeboten. Die entsprechenden Typen heissen LONGINT, bzw. LONGCARD. Der LONGINT-Bereich umfasst alle ganzen Zahlen im Bereich von -2'147'483'648 bis 2'147'483'647, der LONGCARD-Bereich geht von 0 bis 4'294'967'295.

Zahlen-Konstanten, wie zum Beispiel 0 oder -4'009, haben von vornherein keinen bestimmten Typ. Der Compiler weiss lediglich, dass es sich um skalare Zahlen handelt. Erst wenn sie in Rechnungen gebraucht werden oder einer Variablen zugewiesen werden, erhalten sie einen Typ. Bei Zuweisungen wird natürlich kontrolliert, ob diese Konstante überhaupt in dieser Variablen Platz hat. In neueren Publikationen wird vereinzelt eine "lange Zahl" mit einem D hinter der Zahl gekennzeichnet., z.B. 12D wäre die LONGCARD Konstante 12. Dies ist in Amiga Modula-2 *nicht* zulässig, aber auch nicht nötig.

In diesem Zusammenhang sei eine vor allem für Numeriker wichtige Anmerkung zu den Operatoren `DIV` und `MOD` erwähnt. Die mathematische Definition des Rests und der Division lautet:

```
x DIV y = q    q≤x/y
x MOD y = r,   0≤r<y für y>0; y<r≤0 für y<0
```
dabei muss immer gelten: `q * y + r = x.`

---

[1] Das British Standard Institute (BSI) und die International Standard Organization (ISO) arbeiten zur Zeit an einem Modula-2 Standard.

Diese Bedingungen sind bei den meisten Implementationen *nicht* erfüllt. So ist bei negativen Divisoren das Resultat der ganzzahligen Division grösser als der Quotient (näher bei Null) und der Rest hat das Vorzeichen des Dividends statt des Divisors. Die Operatoren `DIV` und `MOD` führen in Amiga Modula-2 die numerisch sauberen Operationen durch. Wird jedoch einmal die alte Definition benötigt, können die Operatoren "/" und `REM` verwendet werden. Der "/"-Operator hat bei Zahlen des Typs REAL, LONGREAL oder FFP die gewohnte Bedeutung der Division.

Um die Unterschiede deutlich vor Augen zu führen, soll folgendes Beispiel dienen:

```
WriteString("   i:      DIV 3  MOD 3");
WriteString("  DIV -3 MOD -3");
WriteLn;
FOR i = 4 TO -4 BY -1 DO
 WriteInt(i,4); WriteString(":  ");
 WriteInt(i DIV 3, 6); WriteInt(i MOD 3, 6);
 WriteInt(i DIV -3, 6); WriteInt(i MOD -3, 6);
 WriteLn;
END;
```

ergibt folgende Ausgabe:

```
   i:       DIV 3  MOD 3  DIV -3 MOD -3
   4:         1      1     -2      -2
   3:         1      0     -1       0
   2:         0      2     -1      -1
   1:         0      1     -1      -2
   0:         0      0      0       0
  -1:        -1      2      0      -1
  -2:        -1      1      0      -2
  -3:        -1      0      1       0
  -4:        -2      2      1      -1
```

Werden im Gegensatz dazu die Operatoren der alten Definition, "/" und "REM" verwendet, ergibt sich folgendes:

```
WriteString("   i:       /  3  REM 3");
WriteString("   /  -3 REM -3");
WriteLn;
FOR i = 4 TO -4 BY -1 DO
 WriteInt(i,4); WriteString(":  ");
 WriteInt(i / 3, 6); WriteInt(i REM 3, 6);
 WriteInt(i / -3, 6); WriteInt(i REM -3, 6);
 WriteLn;
END;
```

ergibt folgende Ausgabe:

```
 i:       /  3  REM 3   /  -3 REM -3
 4:        1     1     -1     1
 3:        1     0     -1     0
 2:        0     2      0     2
 1:        0     1      0     1
 0:        0     0      0     0
-1:        0    -1      0    -1
-2:        0    -2      0    -2
-3:       -1     0      1     0
-4:       -1    -1      1    -1
```

Dabei fällt vor allem die Symmetrie bei der alten Definition auf. Die korrekte Lösung hat keine Symmetrie, ist aber kontinuierlich.

Gleitpunkt-Zahlen

Zum Rechnen mit Gleitpunkt-Zahlen seien folgende Anmerkungen gemacht: der REAL-Bereich umfasst Zahlen von –1.0E38 bis 1.0E38. Da dieser Bereich oft nicht genügt, gibt es in Amiga Modula-2 auch noch den Typ LONGREAL, der von –1.0E308 bis 1.0E308 reicht. Intern werden diese Zahlen nach dem Standard von IEEE abgelegt. Motorola entwickelte eine dritte, spezielle Darstellung der Gleitpunkt-Zahlen, das FFP (fast floating point) Format. Auch dieses Format wird vom Amiga Modula-2 Compiler unterstützt (nähere Details entnehme man dem Kapitel SYSTEM). Es umfasst zwar einen kleineren Bereich als die REAL Darstellung (es sind lediglich Zahlen von –1.0E19 bis 1E19 darstellbar), dafür sind die Operationen etwas schneller.

Bei den Konstanten einer Gleitpunkt-Zahl gelten im Wesentlichen dieselben Regeln wie im skalaren Fall: von vornherein haben die Zahlen keinen bestimmten Typ. Er wird erst bei Rechnungen oder Zuweisungen ermittelt und geprüft. Es gibt also auch hier keine Unterscheidung mittels eines D in der Schreibweise, wie dies in einigen Publikationen zu finden ist: 1.234D201 ist nicht erlaubt. Stattdessen schreibt man einfach 1.234E201.

Restriktionen des Einpass-Compilers

Da der Compiler in einem Durchlauf den Code erzeugt, müssen alle referenzierten Objekte bereits bekannt sein. Eine Ausnahme bilden Zeigerdeklarationen, die auch vor der entsprechenden Struktur deklariert werden können. Muss eine Prozedur aufgerufen werden, die erst später deklariert wird, muss diese Prozedur vorwärtsdeklariert werden. Dazu wird der Prozedurkopf geschrieben, und anstelle des Prozedurrumpfs steht das reservierte Wort `FORWARD`. Die Prozedurimplementation geht dann unverändert vor sich. Beispiel:

```
PROCEDURE ForwProc(i: INTEGER); FORWARD;
...
  (* Aufruf von ForwProc innerhalb einer Prozedur *)
...
PROCEDURE ForwProc(i: INTEGER);
BEGIN
  (* Anweisungen *)
END ForwProc;
```

Anordnung (Alignment)

In Amiga Modula-2 werden Typen bis auf Bytegrenzen gepackt. Einige Beispiele sollen dies verdeutlichen:

```
TYPE
  Tag = (mon, die, mit, don, fre, sam, son);
  Temperatur = [-50..100];
  Set = SET OF [0..7];
```

Alle oben genannten Typen nehmen nur ein Byte in Anspruch. Typen, mehr als ein Byte beanspruchen, beginnen immer an Wortgrenzen (MC68000-Anforderung).

Typentransfer und Typenkonversion

Die systemnahe Programmierung macht es zeitweilig nötig, einer Variable einen neuen Typ zu geben. Dabei müssen jedoch die Grössen des Ursprungtyps sowie des Zieltyps gleich gross sein. Beispiel:

```
VAR
  bs: BITSET;
  card: CARDINAL;
...
bs:=BITSET(card);
```

In diesem Beispiel wird die Bitkombination der Speicherzelle, die für die Variable `card` reserviert ist, unverändert in die Speicherzelle der Variablen `bs` übernommen. Programme, die diesen Mechanismus verwendeten, waren allerdings kaum portierbar, da der Speicherbedarf der einzelnen Typen nicht auf allen Rechnern gleich ist. Wird ein Typentransfer benötigt, muss neu die Prozedur `CAST` aus dem Modul `SYSTEM` importiert werden. Dadurch ist die Systemabhängigkeit angezeigt. Dieser Mechanismus ist zwar bisher noch nicht verwendet worden, ist aber mit Sicherheit in dem in Kürze zu erwartenden Modula-2 Standard enthalten.

Der im Beispiel verwendete Mechanismus ist ebenfalls noch enthalten. Allerdings wird dann kein Typentransfer, sondern eine sichere Typenkonversion ausgeführt. Diese macht eigentlich die Prozeduren `ORD`, `CHR`, `FLOAT` und `VAL` unnötig, sie bleiben jedoch vorderhand aus Kompatibilitätsgründen in der Sprache enthalten, sollten aber nicht mehr verwendet werden. Die Typenkonversion funktioniert nur mit den unstrukturierten Typen (u.a. Unterbereiche und Aufzählungstypen).

`LONGINT("A")`	entspricht	`ORD("A")`
`CHAR(65)`	entspricht	`CHR(65)`
`REAL(100)`	entspricht	`FLOAT(100)`
usw.		
`BITSET(3)`	GEHT NICHT! (nicht unstrukturiert)	

Die Verwendung von Typentransferfunktionen hat den Vorteil, dass der Compiler angewiesen werden kann einen ganz bestimmten Typ zu erzeugen. Die Funktion `ORD()` konvertiert immer zu einem `LONGINT`. Soll aber das Resultat einer Konversion z.B. einer `INTEGER` Variablen zugewiesen werden, muss es anschliessen wieder zu `INTEGER` konvertiert und dazu zusätzlicher Code erzeugt werden.

Sonstiges

In Modula-2 kann ein Modul mit einer Priorität versehen werden. Diese Prioritätsangabe in eckigen Klammern nach dem Modulnamen wird vom Amiga Modula-2 Compiler *ignoriert*.

Die Funktion `HIGH` kann nur auf "offene Feldparameter" angewendet werden. Die Anwendung auf andere Felder ist ohnehin zweifelhaft, da die untere Grenze nicht Null sein muss.

Die Funktionen `SIZE` und `VAL` müssen *nicht* mehr vom Modul SYSTEM importiert werden. Sie sind vordeklarierte Funktionen. `SIZE` gibt die Grösse eines Typs oder einer Variablen in Byte zurück.

Im Gegenzug ist der Typ `BITSET` neu vom Modul SYSTEM zu importieren.

In gewissen Implementationen von Modula-2 ist es üblich, den Platz für eine neue Zeigervariable mit `NEW` und `DISPOSE` zu verwalten. In Amiga Modula-2 gibt es weder `NEW` noch `DISPOSE`. Stattdessen müssen direkt die Prozeduren `ALLOCATE` und `DEALLOCATE` aus `Storage` - oder entsprechende Prozeduren aus `Heap`, `Exec`, `Intuition` - aufgerufen werden. Beispiel:

```
TYPE
  Strukt = RECORD
    a: INTEGER;
    b: CHAR;
    c: INTEGER;
  END;
VAR zeiger: POINTER TO Strukt;
...
ALLOCATE(zeiger, SIZE(zeiger^));
  (* entspricht NEW(zeiger) *)
```

Die Behandlung von Coroutinen ist nicht mehr im Pseudomodul SYSTEM vorhanden. Stattdessen bietet ein herkömmliches Bibliotheksmodul Prozeduren zur Behandlung von Coroutinen (→ Coroutines).

Systemspezifische Erweiterungen / SYSTEM

Nebst den besprochenen Eigenschaften von Amiga Modula-2 gibt es auch einige Änderungen, die von der Anpassung an das Betriebssystem herrühren.

Variablen vom Typ BOOLEAN werden intern *nicht* als [FALSE,TRUE] dargestellt, sondern als 0 und -1. Dies führt zu deutlich dichterem MC68000-Code in Boole'schen Ausdrücken und erspart Umrechnungen nach einem Betriebssystem-Aufruf. Werden Boole'sche Werte als Argument an Standardfunktionen übergeben - beispielsweise ORD - wird vom Compiler automatisch eine Umrechnung übernommen. Der internen Repräsentation ist nur in der systemnahen Programmierung Rechnung zu tragen.

Die Verwendung von Variablen des Typs REAL hat zur Folge, dass zusätzliche Moduln geladen werden. Der Programmierer muss dazu keine besonderen Vorkehrungen treffen, da der Compiler diese Eintragungen vornimmt. Werden Variablen vom Typ LONGREAL benutzt, wird ebenfalls ein zusätzliches Modul geladen. Spielt die Genauigkeit und der Bereich keine Rolle, können auch Variablen des Typs FFP (→ SYSTEM) benutzt werden. Diese Operationen der Zahlen in FFP-Darstellung sind im ROM enthalten, und vergrössern darum die Codegrösse *nicht*.

Parameter werden normalerweise via den Stack an die Prozeduren übergeben. Speziell für Library-Prozeduren wurde ein Mechanismus gefunden, der die Parameterübergabe via Register erlaubt. Es ist möglich, auch in anderen Prozeduren davon Gebrauch zu machen. Die Benutzung ist allerdings nicht ungefährlich, da die Register nicht automatisch gesperrt sind und eventuell bei der Auswertung eines Ausdrucks überschrieben werden.

Registerparameter haben unmittelbar nach dem Namen des formalen Parameters eine Zahl zwischen 0 und 15, eingeschlossen in geschweifte Klammern. Die Datenregister werden durch die Zahlen 0 bis 7 repräsentiert, die Adressregister durch die Zahlen 8 bis 15. Beispiel:

```
PROCEDURE Beispiel(a0{8}: ADDRESS; d0{0}: LONGINT);
```

Amiga Modula-2 besitzt zusätzlich ein reserviertes Wort CODE. Dieses wird ebenfalls hauptsächlich in Library-Moduln verwendet. Statt eines Prozedurkörpers steht das reservierte Wort CODE, gefolgt von einer negativen Zahl. Dies bewirkt das Erzeugen einer einzigen MC68000-Anweisung, nämlich

```
JSR d(A6)
```

Die angegebene Zahl ergibt den Wert von d.

Eine weitere Erweiterung erlaubt es für Amiga Bibliotheken (z.B. Intuition.library) Schnittstellen-Moduln zu schreiben, die dem Compiler erlauben, direkt den Aufrufcode für die Prozeduren zu generieren. Die Bibliothek wird vom Laufzeitsystem dann auch automatisch geöffnet und nach Programmende geschlossen. Um ein DEFINITION MODULE als Schnittstellen-Modul zu definieren, müssen nach dem Modulnamen der Name der Bibliothek und die verlangte Versionsnummer in geschweiften Klammern gesetzt werden.

```
DEFINITION MODULE Intuition{"intuition.library",33};
```

Zu einem Schnittstellen-Modul muss kein Implementationsmodul geschrieben werden. Die Prozeduren müssen jedoch alle eine CODE-

Anweisung enthalten, die ihrem Offset in der Librarytabelle angeben. Der Compiler sorgt im Falle eines Prozeduraufrufs selbständig dafür, dass das A6 Register mit der Adresse der Library geladen wird. Die nachfolgende Anweisung ist dann `JSR d(A6)`, wobei für `d` der angegebene Offset verwendet wird.

Da die Bibliotheken automatisch geöffnet werden, ist es im Gegensatz zu C-Programmen nicht nötig, für die Bibliotheksadresse eine Variable zu deklarieren. Manchmal benötigt man aber doch die Adresse der Bibliothek, beispielsweise um auf die `IntuitionBase` zugreifen zu können. Dies kann mit der Funktion `ADR` geschehen. Dazu importiert man das Modul unqualifiziert und verwendet als Parameter von `ADR` den Modulnamen:

Beispiel:

```
...
IMPORT Intuition;
...
VAR
 intuiBase:IntuitionBasePtr;
...
BEGIN
 intuiBase:=ADR(Intuition);
 ...
```

Das AmigaDOS verwendet neben "normalen" Zeigern sogenannte BCPL-Zeiger. Dies sind Zeiger, deren Wert 4 mal kleiner ist, als die Speicheradresse, auf welche sie zeigen. M2Amiga erlaubt die Deklaration solcher Zeiger. Dies geschieht, indem man einen Zeiger als `BPOINTER TO ...` deklariert. So enthält beispielsweise das Modul Dos die Deklaration

```
FileHandlePtr=BPOINTER TO FileHandle.
```

Variablen von diesem Typ kann man wie normale Zeigervariablen verwenden, der Compiler generiert nötigenfalls automatisch die Multiplikation mit 4, um auf die Elemente der FileHandle zuzugreifen. Wird eine Typenkonversion eines `BPOINTER` nach `ADDRESS` oder umgekehrt durchgeführt, dann wird ebenfalls der Wert des Zeigers entsprechend konvertiert. Zu Beachten: Wird statt einer Typen-

konversion ein Typentransfer mit CAST durchgeführt, wird *nicht* angepasst.

Soll eine Variable zu allen BPOINTER Typen kompatibel sein, muss diese vom Typ BPTR sein. Dieser hat für die BPOINTER-Typen die gleiche Funktion, wie ADDRESS für die POINTER-Typen.

Im folgenden wird das Pseudomodul SYSTEM besprochen. Nicht besonders erklärte Typen oder Prozeduren haben die Eigenschaften, wie sie in jedem Lehrbuch erklärt sind. Änderungen oder Unterschiede sind erklärt.

```
DEFINITION MODULE SYSTEM;
TYPE
  BYTE;
  WORD;
  ADDRESS = POINTER TO BYTE;
  BPTR = BPOINTER TO BYTE;
  BITSET = SET OF [0..15];
  LONGSET = SET OF [0..31];
  FFP;

(*
  T entspricht skalaren Typ mit Grösse <= 4 byte
*)

PROCEDURE ADR    (VAR x   : AnyType): ADDRESS;
PROCEDURE INLINE(     x   : WORD);
PROCEDURE REG    (     reg: [0..15]): LONGINT;
  (* reg const *)
PROCEDURE SETREG(     reg: [0..15];
                      val: T);
  (* reg const.  SIZE(T)=4 *)
PROCEDURE TSIZE  (     AnyType): LONGINT;
PROCEDURE SHIFT  (     x: T;
                      n: LONGINT): T;
PROCEDURE CAST   (     AnyType1;
                      x: AnyType2): AnyType1;

END SYSTEM.
```

Anmerkung: Nicht mehr im Modul SYSTEM sind enthalten:

SIZE neu Standardprozedur

PROCESS, NEWPROCESS, TRANSFER:
Coroutinen-Behandlung neu mit
Bibliotheksprozeduren (→ Coroutines)

Anmerkung: Die Speichereinheit auf dem MC68000, auf welche individuell zugegriffen werden kann, ist das Byte (BYTE). Der Parametertyp ARRAY OF WORD sollte nicht mehr verwendet, und durch ARRAY OF BYTE ersetzt werden.

Der Typ ADDRESS ist mit allen Zeigervariablen, die mit POINTER TO ... deklariert wurden, sowie mit dem Typ LONGINT kompatibel.

Der Typ BPTR ist mit allen Zeigervariablen, die mit BPOINTER TO ... deklariert wurden, sowie mit den Typ LONGINT kompatibel.

Der Typ BITSET entspricht dem bisher vordefiniertem Typ BITSET. Da BITSET ein Wort gross sein soll, ist dieser Typ maschinenabhängig, warum er nun vom Modul SYSTEM importiert werden muss. LONGSET entspricht einem Set in "Langwort"-Grösse.

FFP ist der MC68000-spezifische Typ der Gleitpunktzahlen. Diesem Typ können beliebige konstante Gleitpunktzahlen zugewiesen werden, und die gleichen mathematischen Grundfunktionen wie für den Typ REAL sind für diesen Typ definiert.

Die Prozedur INLINE fügt die übergebenen Worte (es sind beliebig viele erlaubt) unverändert in den Code ein. Es ist Ihnen überlassen, dass die übergebenen Daten einen vernünftigen Code ergeben.

Die Funktion REG gibt den Wert des angesprochenen Registers zurück. Der Parameter reg muss ein konstanter Ausdruck sein, dessen Wert zwischen 0 und 15 liegt. Die Datenregister D0 bis D7 werden durch die

Werte 0 bis 7 angesprochen, die Adressregister A0 bis A7 durch die Werte 8 bis 15.

Die Prozedur SETREG legt den angegebenen Wert im angegebenen Register ab. Für den Parameter reg gilt das gleiche wie bei REG. Das angesprochene Register wird dadurch nicht markiert, d.h. der nächste Ausdruck kann den Registerinhalt bereits wieder zerstören!

Die Prozedur TSIZE erlaubt *keine* Angabe von Kennfeldern (tag fields) um die Grösse einer Variante zu ermitteln. Diese Prozedur sollte ohnehin nicht mehr verwendet werden, da die Definition von SIZE als Parameter neu auch Typen zulässt.

Die Prozedur SHIFT verschiebt die Bitkombination des angegebenen Werts um soviele Stellen wie in n vermerkt. Ein positiver Wert von n verschiebt nach links, ein negativer Wert nach rechts.

Die Prozedur CAST führt einen Typentransfer durch. Dabei wird der Typ der übergebenen Variable an dieser Stelle umgeschaltet. Der Ursprungstyp und der Zieltyp müssen beide den gleichen Speicherbedarf haben. Mit der CAST-Anweisung kann die Typenkontrolle umgangen werden, ihr Einsatz sollte daher möglichst beschränkt bleiben.

9 Das Laufzeitsystem

Ein- und Ausgangscode 3

Compilerunterstützung 4

Die Behandlung von Laufzeitfehlern 5

Unterbrechungsmöglichkeit 6

Die Abschlussprozeduren 7

Die Debugprozeduren 9

Das Konzept der Aktivierungsstufen (Levelkonzept) 10

Fehlermeldungen des Laufzeitsystems 12

- Prozessor Trap 12
- ^C (Benutzer Unterbruch) 13
- Fehler beim Öffnen der xxx.library 13
- Programmierter Halt 13
- Ungültiger Case-Index 13
- Funktion ohne RETURN beendet 14
- Stapelüberlauf 14
- Ungültiger Aufruf von Arts.Call 14
- Adress-Fehler 15
- Ungültige Instruktion 15
- Division durch Null 15

Bereichsfehler 16
Überlauf 17

Das Laufzeitsystem bildet die Basis für jedes Programm, das mit M2Amiga entwickelt wurde. Es steht dem Programm während dessen Ausführung zur Seite und übernimmt im Fehlerfall die Kontrolle. Das Laufzeitsystem ist in mehrere Teilbereiche unterteilbar. Diese Teilbereiche sind jedoch eng ineinander verflochten und deshalb in einem Modul realisiert. Dieses Modul heisst Arts (Amiga Runtime System, oder eben das Laufzeitsystem).

Diese Teilbereiche können weiter in zwei Kategorien eingeteilt werden. Der ersten Kategorie werden alle Teilbereiche zugeteilt, die für den Programmablauf unerlässlich sind. Dies sind der Ein- und Ausgangscode sowie die Compilerunterstützung. Die zweite Kategorie umfasst die restlichen Teilbereiche, nämlich die Behandlung der Laufzeitfehler und Unterbrechungen, die Abschluss- und Debugprozeduren.

Ein- und Ausgangscode

Ein- und Ausgangscode (Entry/Exit Code) bilden eine Klammer um das Anwenderprogramm und verbinden es mit dem Betriebssystem.

Der Eingangscode vervollständigt die Laufzeitumgebung und initialisiert die Programmumgebung. In dieser Phase werden die folgenden Schritte ausgeführt:

- Übernahme der Programmargumente von der Betriebssystemumgebung.

  Bei dieser Übernahme müssen die Unterschiede des Aufrufs von CLI und Workbench berücksichtigt werden. Die übernommenen Argumente werden in den globalen Variablen von Arts abgelegt und stehen dort dem Programm zur Verfügung. Das Bibliotheksmodul Arguments extrahiert aus diesen Variablen die angeforderten Werte.

- Öffnen aller verwendeten Amiga Bibliotheken.

  Dieses automatische Öffnen ist eine der Spezialitäten von M2Amiga. Alle vom Programm verwendeten Amiga

Bibliotheken werden hier eröffnet. In anderen Programmierumgebungen auf dem Amiga ist dies Aufgabe des Programmierers.

Dieses selbständige Öffnen bringt einige Vorteile: Einerseits kann weder das Öffnen, noch der Test auf dessen Erfolg vergessen werden und andererseits entspricht dies eher der Modula-2 Philosophie, dass Importiertes auf jeden Fall benützt werden kann, ohne Initialisierungs-Routinen aufrufen zu müssen.

- Installation der Prozeduren zur Behandlung von Laufzeitfehlern.

  Damit Laufzeitfehler keine "Guru Meditation" auslösen, muss eine Prozedur zur Behandlung der Laufzeitfehler in der Task-Struktur installiert werden. Eine ausführliche Beschreibung dieser Prozedur folgt weiter unten.

  Die Behandlungsprozedur für Benutzerunterbrüche durch <CTRL-C> oder den Dos Befehl 'BREAK' wird ebenfalls hier installiert.

Diese drei Schritte bereiten die Programmumgebung so vor, dass das Programm nun aufgerufen werden kann. Es ist sichergestellt, dass - im Normalfall - das Programm am Ende den Ausgangscode ausführen wird.

Der Ausgangscode hat die Aufgabe alle Installationen des Eingangscodes rückgängig zu machen, die Amiga Bibliotheken zu schliessen und die Kontrolle an die aufrufende Instanz, also das CLI oder die Workbench, zurückzugeben.

Compilerunterstützung

Die Compilerunterstützung umfasst eine Reihe von Prozeduren und Funktionen, die von Programmen verwendet werden, obwohl sie nicht ausdrücklich importiert wurden. Diese Prozeduren sind zu komplex um

sie direkt bei jeder Verwendung in das Programm einzukodieren. Anstelle wird ein Aufruf in das Laufzeitsystem erzeugt.

In unserem Fall beschränkt sich die direkte Compilerunterstützung auf Prozeduren zur Multiplikation und Division von 32-bit Datenwerten, sowie auf die Stapelüberlaufkontrolle und auf die Anzeige von Laufzeitfehlern.

Für das folgende Programmstück wird demnach nicht direkt der Maschinencode der Multiplikation, sondern nur ein Prozeduraufruf in das Laufzeitsystem erzeugt.

```
VAR
 x,y,z: LONGINT;
BEGIN
 z:=x*y;
 (*
  * Dies entspricht exakt der Anweisung
  * z:=Arts.Muls32(x,y);
  *)
END
```

Die Behandlung von Laufzeitfehlern

Viele Fehler in einem Quellentext können bereits zur Compilationszeit eines einzelnen Moduls ausfindig gemacht werden. Dabei sind nicht nur syntaktische Fehler wie vergessene Strichpunkte oder unausgeglichene Klammerpaare eingeschlossen. Darüber hinaus können etwa fehlende Deklarationen, typeninkompatible Vergleiche oder Zuweisungen, Namenskonflikte und dergleichen mehr festgestellt werden. Selbst eine Division mit der Konstanten 0 (Null) wird vom Compiler beanstandet.

Leider gibt es noch einige Fehlersituationen die der Compiler nicht zur Compilierzeit erkennen kann. Solche Fehler treten zur Laufzeit des Programms auf, weshalb sie Laufzeitfehler genannt werden.

Die Laufzeitfehler die bei einem M2Amiga Programm auftreten können, werden durch einen Fehlerzustand des Programmes erzeugt. Diese

Fehlerzustände werden entweder vom Prozessor oder vom Programm selbst erkannt.

Auf dem Amiga dürfte wohl allen die Reaktion des Betriebssystems auf Laufzeitfehler bekannt sein: die "Guru Meditation". Diese erscheint dann, wenn das Betriebssystem durch einen fehlerhaften Task (Prozess) so empfindlich gestört wurde, dass als Lösung nur der Neustart des Systems bleibt.

Eine Quelle für den (so hartnäckig) meditierenden Guru sind die Traps des Prozessors. Sobald der Prozessor einen ungültigen Zustand entdeckt, wird zur Fehlerbehandlung eine entsprechende Trap-Prozedur aufgerufen. Die Gründe für diese Prozessor-Traps sind unter anderem, die Division durch Null, ein Fehler bei der Adressierung einer Variablen oder ein Bereichsfehler, der durch die "CHK" Instruktion erkannt wurde. Da insbesondere die Instruktion "CHK" vom Compiler zur Bereichskontrolle erzeugt wird, muss zur Vermeidung der "Guru Meditation" die Behandlung der Traps vom Laufzeitsystem übernommen werden.

Die Laufzeitfehler, die vom Programm erkannt werden, werden durch Aufrufe von Prozeduren des Laufzeitsystems angezeigt. Diese Aufrufe können einerseits vom Compiler selbständig erzeugt, oder vom Programmierer ins Programm eingefügt werden. Alle Möglichkeiten zur Erzeugung von Laufzeitfehlern durch den Programmierer sind aus der Modula-2 Anweisung "HALT" entstanden. Die Anweisung "HALT" ist dafür gedacht, um im Fehlerfall ohne Ausweg das Programm zu beenden.

Neben der Anweisung "HALT" bietet das Laufzeitsystem dem Programmierer die Prozeduren Assert, BreakPoint, Error und Requester an. Diese sind im Abschnitt "Arts" im Bibliotheken-Kapitel genauer beschrieben.

Unterbrechungsmöglichkeit

Das Laufzeitsystem von M2Amiga bietet die Möglichkeit, Programme mittels <CTRL-C> fast jederzeit zu unterbrechen. Dieser Unterbruch ist unabhängig vom Programmablauf, kann also asynchron eintreffen.

Bei diesen asynchronen Unterbrüchen (Interrupts) muss einiges kontrolliert werden, um einen störungsfreien Abbruch zu ermöglichen. Diese Unterbrüche können nämlich willkürlich innerhalb einer Systemfunktion ausgelöst werden. So dürfen keinerlei Unterbrüche während der Ausführung einer Dos-Funktion akzeptiert werden. Dies wird in Arts entsprechend abgefangen, d.h. der Unterbruch innerhalb einer Dos-Funktion wird nicht zu Ende geführt. Daneben gibt es weitere Situationen in denen ein Unterbruch fatale Folgen haben kann. Leider können diese nicht klar erkannt werden und führen deshalb zu Problemen beim Abbruchversuch. Im Normalfall wird nach einem Unterbruchswunsch (Tastenkombination <CTRL-C> oder BREAK-Befehl [->DOS]) getestet ob gerade eine Dos-Operation ausgeführt wird. Ist dies der Fall, wird der Unterbruch verwehrt und das Programm läuft unverändert weiter. Wird keine Dos-Operation ausgeführt, erscheint der Abbruch-Requester.

Die Abschlussprozeduren

Der Modulrumpf eines Modula-2 Programms hat die Aufgabe, einen definierten Zustand zu schaffen. Oft werden dabei Betriebsmittel reserviert, beispielsweise wird Speicher alloziert. Beim Programmende besteht für das Modul keine Möglichkeit, die Betriebsmittel wieder freizugeben. Da das Betriebssystem des Amiga die Betriebsmittel zwar vergibt, aber nicht kontrolliert, bleiben diese auch nach Programmende reserviert. Daher musste ein Mechanismus gefunden werden, der die Freigabe nach Programmende ermöglicht. Dies gilt besonders für fehlerhafte Programme, bei denen der Programmabbruch an einer beliebigen Stelle möglich ist.

Das Laufzeitsystem von Amiga Modula-2 bietet dafür das Konzept der Abschlussprozeduren (termination procedures). Ein Modul kann beliebige parameterlose Abschlussprozeduren installieren, welche dann im Fall eines Programmabbruchs oder am Programmende aufgerufen werden.

PROCEDURE TermProcedure(proc: PROC);

Die Prozedur proc wird in die Liste der Abschlussprozeduren eingetragen. Am Programmende wird die zuletzt eingetragene Prozedur als erste aufgerufen. Damit ist gewährleistet, dass die Abschlussprozeduren tieferer Module nicht vor denen des Hauptprogrammes aufgerufen werden.

PROCEDURE RemoveTermProc(proc: PROC);

Die zuvor installierte Abschlussprozedur wird aus der Liste der Abschlussprozeduren wieder entfernt. Beim Programmende werden alle zum Programm gehörenden Abschlussprozeduren selbständig entfernt.

Beispiel:

```
MODULE Terminal;
FROM Arts IMPORT Assert,TermProcedure;

CONST window="CON:0/0/640/100/out";
VAR out:FileHandlePtr;

PROCEDURE Cleanup;
BEGIN
 Close(out)
END Cleanup

BEGIN (* Rumpf *)
 out:=Open(ADR(window),newFile);
 Assert(out#NIL,ADR("Öffnen-Fehler"));
 TermProcedure(Cleanup)
END Terminal.
```

Bei der Initialisierung des Moduls `Terminal` wird ein Fenster eröffnet, auf das über `out` zugegriffen werden kann. Beim Programmende schliesst die als Abschlussprozedur übergebene Prozedur `Cleanup` das Fenster wieder. Nach diesem Mechanismus schliesst das Modul "FileSystem" alle offenen Dateien, und "Heap" sowie "Storage" deallozieren alle zuvor allozierten Speicherbereiche. Fehlerhafte Programme verhindern damit das Entstehen von "unsauberen" Situationen und ersparen einen Neustart des gesamten Systems.

Die Debugprozeduren

Neben den Abschlussprozeduren können auch Prozeduren installiert werden, die bei der Fehlersuche eine Hilfe anbieten. Diese parameterlosen Prozeduren können beim Auftreten eines Laufzeitfehlers durch Anklicken des "debug" Gadgets aktiviert werden.

PROCEDURE DebugProcedure(debugger: PROC);

> Die übergebene Prozedur wird als Debugger installiert. Falls ein Laufzeitfehler auftritt, wird diese Prozedur auf den Wunsch des Anwenders hin aufgerufen. Diese Prozedur sollte sich darauf beschränken den aktuellen Zustand von Variablen anzuzeigen.

PROCEDURE RemoveDebugProc(debugger: PROC);

> RemoveDebugProc entfernt die Prozedur debugger wieder. Dies geschieht selbständig wenn das Programm abgebrochen oder beendet wird.

Das folgende Beispiel zeigt wie mit BreakPoint und DebugProcedure der Ablauf des Programms kontrolliert werden kann.

Beispiel:

```
MODULE debuggerDemo;

 FROM SYSTEM IMPORT
  ADR;
 FROM Arts IMPORT
  BreakPoint,DebugProcedure;
 FROM InOut IMPORT
  WriteInt,WriteLn,WriteString;

 VAR
  x,y,z: INTEGER;

 PROCEDURE Debug;
 BEGIN
  WriteString(
   "Die Variablen haben folgende Werte:"
```

```
  );
  WriteLn;
  WriteString("x = ");
  WriteInt(x,5);
  WriteString("  y = ");
  WriteInt(y,5);
  WriteString("  z = ");
  WriteInt(z,5);
  WriteLn
END Debug;

BEGIN
  DebugProcedure(Debug);
  x:=10;
  FOR y:=3 TO 0 BY -1 DO
    BreakPoint(ADR("VOR z:=x/y"));
    z:=x/y
  END
END debuggerDemo.
```

Das Konzept der Aktivierungsstufen (Levelkonzept)

Neben den Prozeduraufrufen von Modula-2 wird in Arts das Konzept der Aktivierungsstufen eingeführt. Eine Aktivierungsstufe ist eine Erweiterung des normalen Prozeduraufrufs. Als neue Stufe aktiviert, erhält eine Prozedur die Eigenschaften eines eigenständigen Programmes. Dies umfasst vor allem die Behandlung von Fehlerzuständen und der Programmtermination. Alle Module die über allozierte Betriebsmittel Buch führen, notieren sich auch die Stufe von der aus sie alloziert wurden. Die Betriebsmittel werden erst dann freigegeben wenn die entsprechende Aktivierungsstufe abgebaut wird.

Dies ist die Grundlage für das Aufstarten von Programmen im M2Amiga Loader. Ein Programm das andere Programme aufruft, muss in seiner Einheit bestehen bleiben. Falls das aufgerufene Programm fehlerhaft abbricht, so müssen ebenfalls Betriebsmittel des aufgerufenen Programms wieder frei gegeben werden.

PROCEDURE Call(proc: PROC);

Call ruft die angegebene Prozedur auf der nächsthöheren Aktivierungsstufe auf.

PROCEDURE CurrentLevel(): INTEGER;

CurrentLevel gibt die aktuell Aktivierungsstufe an. Diese wird zur Zuordnung von Betriebsmitteln zu den einzelnen Aktivierungsstufen verwendet.

PROCEDURE Terminate(level: INTEGER);

Terminate beendet die angegebene Aktivierungsstufe unmittelbar. Falls die übergebene Stufenangabe nicht gültig ist oder Null ist, wird das gesamte Program abgebrochen.

Beispiel:

```
MODULE levels;
(*
 * Demonstration verschiedener
 * Aktivierungsstufen
 *)
FROM Arts IMPORT
 Call, CurrentLevel, TermProcedure;
FROM InOut IMPORT
 WriteInt,WriteLn,WriteString;

PROCEDURE TermProc1;
BEGIN
 WriteString(
  "Abschluss Prozedur 1: Stufe ist "
 );
 WriteInt(CurrentLevel(),3);
 WriteLn
END TermProc1;

PROCEDURE TermProc2;
BEGIN
 WriteString(
  "Abschluss Prozedur 2: Stufe ist "
 );
 WriteInt(CurrentLevel(),3);
```

```
 WriteLn
END TermProc2;

PROCEDURE Level2;
BEGIN
 TermProcedure(TermProc2);
 WriteString("Prozedur Level2: Stufe ist ");
 WriteInt(CurrentLevel(),3);
 WriteLn
END

BEGIN
 TermProcedure(TermProc1);
 WriteString("Hauptprogramm 1: Stufe ist ");
 WriteInt(CurrentLevel(),3);
 WriteLn;
 WriteString("Call(Level2)"); WriteLn;
 Call(Level2);
 WriteString("Hauptprogramm 2: Stufe ist ");
 WriteInt(CurrentLevel(),3);
 WriteLn;
END levels.
```

Fehlermeldungen des Laufzeitsystems

Das Laufzeitsystem meldet verschiedene Ereignisse, die ein Weiterlaufen des Programms verunmöglichen, mittels eines Requesters dem Benutzer. Der Benutzer kann wählen, ob das Program abgebrochen oder "debuggt" werden soll.

Prozessor Trap

Im Programm wurde eine TRAP Instruktion [MC68000] angetroffen. Trap-Instruktionen werden vom Compiler nicht erzeugt (ausser TRAP 14 und 15, siehe unten), aber sie können mit INLINE eingefügt werden. Der Fehler kann auch auftreten, wenn wegen einer nicht initialisierten Prozedurvariable eine Speicherstelle angesprungen wird, die keinen gültigen Code enthält.

Eine Ausnahme bilden die TRAP Instruktionen TRAP 14 und TRAP 15. Diese werden vom Compiler für die Bereichs- bzw. Überlaufprüfung verwendet. Diese Trapinstruktionen führen deshalb nicht zu einer

Prozessor-Trap Meldung, sondern werden als "Bereichsfehler (TRAP 14)" bzw. "Überlauffehler (TRAP 15)" dargestellt.

^C (Benutzer Unterbruch)

Dieser Requester erscheint, wenn das Programm mit CTRL-C oder dem Break-Befehl [DOS] unterbrochen wurde. Man hat zwei Möglichkeiten zur Wahl: Mit "abbrechen" kann man das Programm beenden; mit "weiter" kann das Programm fortgeführt werden. Letzteres ist nützlich, wenn man voreillig die CTRL-C Taste gedrückt hat.

Fehler beim Öffnen der xxx.library

Da das Laufzeitsystem alle Libraries selbstständig öffnet, ist es dem Benutzer unmöglich zu testen, ob die Library erfolgreich geöffnet werden konnte. Ist dies nicht der Fall, erscheint diese Fehlermeldung. Das Programm könnte ja ohnehin nicht ausgeführt werden.

Programmierter Halt

Tritt auf, wenn die Modula-2 Anweisung "HALT" abgearbeitet wurde.

Ungültiger Case-Index

Diese Fehlermeldung erhalten Sie dann, wenn der Wert des getesteten Ausdrucks zwischen dem kleinsten und grössten CASE-Label ist, aber kein entsprechendes Label vorhanden ist.

Beispiel:

```
i:=2;
CASE i OF  (* getesteter Ausdruck: i *)
| 1: Write('1');  (* kleinstel Label: 1 *)
| 3: Write('3');  (* grösstes Label: 3 *)
END;
```

Selbstverständlich verunmöglicht ein radikales Verwenden von ELSE-Klauseln (innerhalb CASE) diesen Laufzeitfehler. Dies erschwert jedoch

die Fehlersuche, da unvorhergesehene (unmögliche?) Fälle nicht unmittelbar auffallen.

Falls der Wert des getesteten Ausdrucks *nicht* zwischen dem kleinsten und grössten CASE-Label liegt, wird das Programm mit einem Bereichsfehler abgebrochen.

Funktion ohne RETURN beendet

Funktionsprozeduren müssen durch eine RETURN-Anweisung beendet werden (bei Prozeduren ist diese nicht zwingend). Ist das Ende einer Funktionsprozedur (END-Anweisung) erreicht, ohne dass eine RETURN-Anweisung ausgeführt wurde, so erscheint dieser Fehler. Setzen Sie eine RETURN-Anweisung ein, um ihn zu beheben.

Stapelüberlauf

Jeder Prozeduraufruf belegt einen gewissen Platz auf dem Stapel (Stack). Dort sind Rücksprungadresse, Verweis auf die aufrufende Prozedur, Parameter und lokale Variablen abgelegt. Jedem Task ist nur ein begrenzter Stapelbereich zugeordnet. Ist der verbleibende Stapelbereich zu klein, dann kann keine weitere Prozedur aufgerufen werden, was durch diesen Laufzeitfehler angezeigt wird. Vergewissern Sie sich, dass das Programm fehlerfrei ist (insbesondere bei Rekursion) und vergrössern Sie nötigenfalls den Stapelbereich.

Die Anfangsgrösse des Stapelbereichs kann im CLI mit dem STACK-Befehl [DOS] eingestellt werden. In der Workbench ist die Anfangsgrösse für jedes Programm (Tool) in dessen .info Datei abgelegt. Diese Grösse kann angezeigt und geändert werden mit der Funktion "Info" des Workbench Menus.

Ungültiger Aufruf von Arts.Call:

Arts kann maximal 8 Levels verarbeiten. Ein Aufruf von Call auf dem 8. Level bewirkt diesen Fehler.

Adress-Fehler:

Ein Adress-Fehler wird direkt durch den Prozessor dann ausgelöst, wenn dieser auf ein Wort, ein langes Wort (32 Bit) oder eine Instruktion an einer ungeraden Adresse (Byte-Adressen!) zugreifen will. Dieser Fehler bedeutet meist, dass auf nicht-initialisierte Zeiger oder Prozedurvariablen zugegriffen wurde.

Ungültige Instruktion:

Als ungültige Instruktion wird jedes Wort-Bitmuster bezeichnet, das nicht dem Bitmuster des ersten Worts eines gültigen MC68000-Befehl entspricht. Davon gibt es eine Ausnahme: Das Wort 4AFCH (0100101011111100) entspricht dem "gültigen" Befehl "ILLEGAL". Es ist das einzige Bitmuster, das auch nach zukünftigen Erweiterungen der MC68000-Familie "ILLEGAL" bleibt. Dieser Fehler, bedeutet meist, dass auf nicht-initialisierte Zeiger oder Prozedurvariablen zugegriffen wurde.

Division durch Null:

Zahlen beliebigen Typs können laut Regeln der Mathematik nicht durch Null (0) dividiert werden. Wird in einer Anweisung versucht, durch 0 zu dividieren, kann das Programm nicht weiter ausgeführt werden. Gleich verhält es sich mit der Modulo-Operation (Rest), deren Wert sich aus dem Rest der vorangehenden Division ergibt.

Beispiel:

```
VAR
 i,j:INTEGER;
 r,s:REAL;
...
i:=0; r:=0.0;
(*
 * Folgende Anweisungen führen zum
 * beschriebenen Laufzeitfehler
 *)
j:=j DIV i;
j:=j / i;
j:=j MOD i;
j:= j REM i;
s := s / r;
```

Das Beispiel gilt selbstverständlich auch für die anderen numerischen Typen.

Bereichsfehler:

Bei jeder Zuweisung an einen Unterbereich, bei jedem Zugriff auf ein Feldelement (ARRAY) und vor der Ausführung einer CASE-Anweisung findet eine Prüfung der unteren und der oberen Schranke statt. Bei einer Zuweisung an einen Unterbereich wird geprüft, ob der resultierende Wert in die Zielvariable gespeichert werden kann. Bei einem Zugriff auf ein Feldelement wird geprüft, ob überhaupt ein Element mit dem ausgewerteten Indexausdruck existiert (deklariert ist). Bei einer CASE-Anweisung wird der berechnete Ausdruck als Index in eine Sprungtabelle verwendet, die für jeden der CASE-Label die Adresse der zugehörigen Anweisungen enthält. Es muss also wie bei einem Zugriff auf ein Feldelement überprüft werden ob dies ein gültiger Index ist.

Ist die untere Schranke Null (0) und die obere Schranke kleiner als 2^15 (32768), geschieht diese Überprüfung mit der MC68000-Instruktion "CHK". Die erscheinende Fehlermeldung heisst dann "Bereichsfehler (CHK)". Andernfalls ist die Bereichsprüfung komplexer: Nach erfolgter Prüfung mithilfe von einer oder zwei Vergleichsinstruktionen wird im Fehlerfall (Test des Condition Code Registers) die MC68000-Instruktion "TRAP #14" ausgeführt, wie es auch in der Fehlermeldung ("Bereichsfehler (TRAP 14)") vermerkt ist. Für den Anwender ist diese Unterscheidung allerdings von untergeordneter Bedeutung.

st die untere Schranke Null (0) und die obere Schranke kleiner als 2^15 (32768), geschieht diese Überprüfung mit der MC68000-Instruktion "CHK". Die erscheinende Fehlermeldung heisst dann "Bereichsfehler (CHK)". Andernfalls ist die Bereichsprüfung komplexer: Nach erfolgter Prüfung mithilfe von einer oder zwei Vergleichsinstruktionen wird im Fehlerfall (Test des Condition Code Registers) die MC68000-Instruktion "TRAP #14" ausgeführt, wie es auch in der Fehlermeldung ("Bereichsfehler (TRAP 14)") vermerkt ist. Für den Anwender ist diese Unterscheidung allerdings von untergeordneter Bedeutung.

Beispiel:
```
VAR
 a: ARRAY [0..5] OF INTEGER;
 sub: [0..10];
 i: INTEGER;
...
i := 11;
sub := i;  (* Bereichsfehler, da i>MAX(sub) *)
sub := -i; (* Bereichsfehler, da i<MIN(sub) *)
i := a[i]; (* Bereichsfehler, da kein a[11] *)
```

Überlauf:

Ist das Resultat einer Rechenoperation grösser als es der Maximalwert der beteiligten Typen erlaubt, so tritt Überlauf ein. Die entsprechende Unterschreitung des Minimalwertes wird Unterlauf genannt. Auf Prozessorebene wird Überlauf und Unterlauf hingegen nicht unterschieden, weshalb hier Überlauf hier auch Unterlauf beinhalten soll.

Beispiel:
```
VAR
 i,j,k: INTEGER;
...
i:=30000;
j:=20000;
k:=-30000;
IF (i+j)<0 THEN ... END; (* Überlauf *)
IF (k-j)>0 THEN ... END; (* Unterlauf *)
```

Üblicherweise wird mit der TRAPV-Anweisung auf Überlauf geprüft. Dies ist aber bei einer 16 Bit Multiplikation nicht möglich. Die Über-

laufsprüfung erfolgt hier durch eine explizite Prüfung. Im Fehlerfall erfolgt ein Aufruf der TRAP 15 Instruktion. Im Requester wird angegeben ob eine TRAPV oder TRAP 15 Instruktion ausgeführt wurde.

10 Technische Angaben

Von M2Amiga verwendete Dateiformate 3

Objektdatei 6

Symbol-/Referenzdatei 8

Beispiel 13

Fehlerdatei 17

Laufzeitumgebung von M2Amiga 18

Kontrollcode 21

Dieses Kapitel richtet sich vor allem an den fortgeschritteneren Benutzer von Amiga-Modula-2. Wenn Sie erst in die Programmiersprache Modula-2 einsteigen, können Sie dieses Kapitel überspringen. Wollen Sie Details von Amiga-Modula-2 wissen, so sind Sie hier richtig.

Von M2Amiga verwendete Dateiformate

M2Amiga verwendet 3 spezielle Dateiformate: Objektdatei, Referenz- und Symboldatei sowie Fehlerdatei. Diese drei Formate werden nun ausführlich beschrieben.

Die Dateiformate werden mit Hilfe einer EBNF-ähnlichen Sprache beschrieben. EBNF wird häufig zur Beschreibung der Syntax einer Programmiersprache verwendet (z.B. [PIM3]). Sie ist aber auch geeignet, das Format einer Datei zu beschreiben. Allerdings sind die Terminalsymbole im Falle einer Datei keine Worte (wie z.B. `IF` in Modula-2) sondern entweder Konstanten oder Variablen, die einen bestimmten Typ haben. Aus diesem Grund beschreiben wir Terminalsymbole mit Hilfe von Modula-2 Typen und Konstanten. Hier die Elemente unserer "EBNF":

A=B C. Dies bezeichnet eine Regel, die besagt, dass A durch die Folge BC ersetzt werden soll.

A=B | C. A soll durch B oder C ersetzt werden.

A=[B]. A soll durch B ersetzt oder weggelassen werden.

A={B}. A soll durch eine Folge von 0, 1, 2 oder mehrere B ersetzt werden.

A="ARRAY [0..31] OF CHAR"
A besteht aus 32 Buchstaben.

A="INTEGER(7)".

A besteht aus der Konstanten 7, wobei in der Datei soviel Platz wie für einen INTEGER verwendet wird (2 Byte).

Der Punkt "." am Ende der Regel markiert deren Ende. Alles was nach dem Punkt folgt ist Kommentar. Die oben beschriebenen Regeln können zu komplexeren Regeln zusammengefasst werden, z.B. "A=B C | D E." Dabei bindet " | " schwächer als " ". "A=B C | D E." bedeutet, dass "A" entweder durch "B C" oder durch "D E" ersetzt werden darf. Dies kann mit Klammern umgangen werden: "A=B (C | D) E." Diese Regel besagt, dass "A" durch "B C E" oder durch "B D E" ersetzt werden darf.

Eine vollständige Dateibeschreibung könnte dann so aussehen:

Datei	=	ID NumBlocks {Block} [Name].	<1>
ID	=	"LONGINT(17)".	<2>
NumBlocks	=	"INTEGER". Gibt die Anzahl der folgenden Blöcke.	<3>
Block	=	Size (CodeId | DataId) {Word}.	<4>
Size	=	"LONGINT". Grösse des folgenden Blocks ohne die CodeId bzw. DataId.	<5>
CodeId	=	"INTEGER(1)".	<6>
DataId	=	"INTEGER(2)".	<7>
Word	=	"INTEGER".	<8>
Name	=	"ARRAY [0..7] OF CHAR".	<9>

Die Bedeutung dieser Zeilen im einzelnen:

<1> Die Datei besteht aus einer ID gefolgt von einer Zahl und beliebig vielen (aber auch 0) Blöcken und evtl. einem Namen.

<2> Die Identifikation der Datei hat den Wert 17, und wird in 4 Byte dargestellt.

<3> Die Zahl der Blöcke ist durch zwei Byte gegeben. Mit dem Kommentar wird veranschaulicht, was die EBNF alleine nicht aufzeigen kann; nämlich dass zwischen der Zahl "NumBlocks" und der Anzahl von darauffolgenden Blöcken eine Beziehung besteht. NumBlocks gibt an, wieviele Blöcke folgen.

<4> Ein Block besteht aus einer Grössenangabe und einer Identifikation, gefolgt von beliebig vielen Worten.

<5> Die Grösse des Blocks wird durch vier Byte angegeben. Auch hier wieder ein Kommentar, der genauer die Beziehung zwischen Size und den später folgenden Worten erklärt.

<6> Die CodeId belegt zwei Byte , und hat den Wert 1.

<7> Die DataId belegt ebenfalls zwei Byte und hat den Wert 2.

<8> Das Word besteht aus zwei Byte beliebigen Inhalts.

<9> Der Name besteht aus 8 Zeichen (Byte).

Der folgende Hexdump stellt eine nach obigem Schema gültige Datei dar. Diese enthält die ID. Darauf folgt die Angabe, dass genau ein Block vorhanden ist. Dieser Block hat die Grösse 4. Es ist ein Datenblock und enthält die vier Byte 1E 84 95 0A. Danach folgt noch der Name "Test".

```
00 00 00 11 00 01 00 00 00 04 00 02 1E 84 95 0A
53 65 74 73 00 00 00 00
```

Nach dieser Erklärung des Prinzips der Datenbeschreibung, folgen nun die Dateitypen-Beschreibungen.

Objektdatei

Dieser Dateityp wird vom Compiler produziert und von Linker als Eingabedatei verwendet. In der Objektdatei wird der vom Compiler für ein Modul produzierte Code abgelegt. Der Linker erwartet die zu linkenden Moduln in diesem Format. Im Gegensatz zum Compiler kann der Linker auch Dateien verarbeiten, die mehr als nur ein Modul beinhalten.

Module	=	{HeaderBlock ConstBlock ImportBlock CodeBlock}.
HeaderBlock	=	CODEFILE ModuleId CodeSize DataSize ConstSize Procs Mods.
ImportBlock	=	{ModuleId | LibraryId}{Empty}.
ConstBlock	=	{ConstWord}{ProcJmp}.
CodeBlock	=	{CodeWord}.
ModuleId	=	(MOD | NOIMP) Name Key.
LibraryId	=	LIB Name PadWord LibVersion.
Empty	=	NONE Name Key. Name und Key werden ignoriert
ProcJmp	=	JMP ProcPtr.
Name	=	"ARRAY [0..31] OF CHAR".
Key	=	"ARRAY [0..2] OF INTEGER".
LibVersion	=	"LONGINT".
ProcPtr	=	"LONGINT".
CodeSize	=	"LONGINT". Die Länge des Code-Blocks in Byte.
DataSize	=	"LONGINT".
ConstSize	=	"LONGINT". Die Länge des "{ConstWord}"-Teil des "ConstBlock" in Byte.
Procs	=	"LONGINT". Die Zahl der Prozeduren in "{ProcedureJmp}" des "ConstBlock".
Mods	=	"LONGINT". Die Zahl der Moduln in "ImportBlock", inklusive den Leereinträgen ("Empty").
ConstWord	=	"INTEGER".
CodeWord	=	"INTEGER".
PadWord	=	"INTEGER(0)".

CODEFILE	=	"LONGINT(1)".
NONE	=	"INTEGER(0)".
MOD	=	"INTEGER(1)".
LIB	=	"INTEGER(2)".
NOIMP	=	"INTEGER(3)".
JMP	=	"INTEGER(4EF9H)". MC68000 Jump Befehl.

Die Objektdatei besteht aus vier Blöcken. Der Headerblock enthält die Identifikation des Moduls sowie die Grössen der verschiedenen Blöcke. Mit dieser Information kann der Loader schon wissen, ob das richtige Modul vorliegt (ModuleId) und wieviel Speicherplatz für den Code- und Datenteil benötigt wird. Die Grösse des Code-Teils berechnet sich aus

$$Grösse_{Code\text{-}Teil} = ConstSize+6*Procs+4+4*Mods+CodeSize.$$

Hierin erkennt man wieder die Struktur des Code-Teils. An der niedrigsten Adresse befinden sich die konstanten Zeichenketten, danach folgt die Prozedur-Sprungtabelle, die Sprungbefehle zu allen exportierten Prozeduren enthält (je 6 Byte lang). Danach folgt der Zeiger auf die eigenen globalen Daten (4 Byte) und die Zeiger auf die importierten Moduln (je 4 Byte). Schliesslich folgt der eigentliche Code.

Der Linker produziert aus dieser Objektdatei "Hunks" (→ [DOS]), die in der gebundenen Datei abgelegt werden. Ein "bss_hunk" enthält die Grösse der globalen Daten und ein "data_hunk" den aufbereiteten Codeteil mit der nötigen Relozierungs-Information, um die relativen Adressen der Prozedur-Sprungtabelle in absolute Adressen umzuwandeln und die Modulzeiger richtig zu initialisieren. Weil der Amiga-Loader nur in der Lage ist, Adressen relativ zu einem Hunkanfang zu relozieren, sind die Prozeduradressen in der Objektdatei relativ zum Anfang des Codeteils abgelegt.

Noch eine Bemerkung zum Importblock. Dieser kann bis zu drei Leer-Einträge enthalten. Diese sind reserviert für die Moduln MathReal, MathFFP und MathIEEEDoubBas. Diese Moduln werden vom Compiler automatisch "importiert", wenn Berechnungen mit FFP-, REAL- oder LONGREAL-Zahlen dies erfordern. Dieser "Import" erfolgt

aber erst, nachdem der Compiler schon Code generiert hat, nämlich beim ersten Auftreten einer solchen Operation. Da die Zugriffe auf die konstanten Zeichenketten (zur Laufzeit) relativ zum Programmzähler (PC) erfolgen, muss die Grösse der Modulliste - die sich ja zwischen Code und Konstanten befindet - vom Beginn der Kompilation an festgelegt sein. Aus diesem Grund wird der Platz für diese Moduln auf alle Fälle reserviert.

Symbol-/Referenzdatei

Die Symboldatei wird vom Compiler erzeugt und auch von diesem verwendet. Sie enthält in codierter Form alle wichtigen Angaben eines Definitionsmoduls.

Die Referenzdatei wird vom Compiler erzeugt und vom Debugger verwendet. Sie enthält die vom Debugger benötigten Informationen über die Lage der Variablen im Speicher, sowie die Positionen der Modula-2-Anweisungen innerhalb des Codes.

ReferenceFile	=	REFFILE ModAnchor {ModAnchor} {ObjList ModTag | ObjList ProcTag} ObjList RefTag.
ObjList	=	{Structure | Component | Object | Linkage | PC-Block}.
Structure	=	Enum | Range | Pointer | Set | ProcTyp | FuncTyp | Array | DynArray | Record | Opaque.
Component	=	ParRef | Par | Field.
Object	=	VarRef | Var | Constant | Type | Procedure | Function | Module | Code.
PC-Block	=	PCPos SourcePos.
ModAnchor	=	BYTE(0H) key name modmode [libname]. "libname" erscheint wenn "modmode" den Wert "LIB" hat.
ModTag	=	BYTE(1H) ModNo.
ProcTag	=	BYTE(2H) ProcNo.

RefTag	=	BYTE(3H) adr pno. adr = Adresse, von der an die Variablen des Implementationsmoduls alloziert werden dürfen. pno = Zahl der deklarierten Prozeduren.
Linkage	=	BYTE(4H) StrRef BaseRef. Der früher deklarierte Zeigertyp StrRef soll auf den Typ BaseRef zeigen. Wird benötigt, wenn ein Zeigertyp vor dem Basistyp deklariert wird.
Enum	=	BYTE(10H) size NoConst. size = Zahl der Byte, die für eine Variable von diesem Typ benötigt wird (Dies gilt auch für die nächsten 9 Zeilen). NoConst = Anzahl Elemente in diesem Aufzählungstyp.
Range	=	BYTE(11H) size BaseRef min max sign. BaseRef = Grundtyp dieses Unterbereichtyps. min = Untere Bereichsgrenze. max = Obere Bereichsgrenze. sign = Gibt an, ob es sich um einen vorzeichenbehafteten Typ handelt oder nicht.
Pointer	=	BYTE(12H) size.
Set	=	BYTE(13H) size BaseRef. BaseRef = Elementtyp dieses Mengentyps.
ProcTyp	=	BYTE(14H) size.
FuncTyp	=	BYTE(15H) size ResRef. ResRef = Resultatstyp dieser Prozedur.
Array	=	BYTE(16H) size ElemRef IndexRef. ElemRef = Elementtyp dieses Arrays. IndexRef = Indextyp dieses Arrays.
DynArray	=	BYTE(17H) size ElemRef. ElemRef = Elementtyp dieses Arrays.
Record	=	BYTE(18H) size.
Opaque	=	BYTE(19H) size.
BPointer	=	BYTE(1AH) size.

ParRef	=	BYTE(20H) StrRef parmode name. StrRef = Typ des Parameters. parmode = 0 falls es sich um einen Parameter handelt, der auf dem Stack übergeben wird, 1..17 falls es sich um einen Parameter handelt der in einem Register übergeben wird (1..8 = D0..D7, 9..17 = A0..A7). name = Name des Parameters. Dieser Name ist leer, wenn das Definitionsmodul mit der Option N- oder M- compiliert wurde, oder falls es sich um ein Schnittstellen-Modul handelt.
Par	=	BYTE(21H) StrRef parmode name. wie ParRef.
Field	=	BYTE(22H) StrRef offset name. StrRef = Typ dieses Record-Feldes offset = Adresse des Feldes relativ zum Anfang des Records. name = Name des Feldes.
VarRef	=	BYTE(30H) StrRef level address varmode name exported. StrRef = Typ dieser Variable. level = Schachtelungstiefe der Prozedur, in der diese Variable deklariert wurde. 0=global. address = Adresse dieser Variable. varmode = siehe parmode bei ParRef. name = Name der Variable. exported = gibt an, ob Variable exportiert wird.
Var	=	BYTE(31H) StrRef level address varmode name exported. siehe VarRef
Constant	=	BYTE(32H) StrRef ModRef value name exported. StrRef, name und exported siehe VarRef. ModRef = Nummer des Moduls, in der diese Konstante deklariert wurde. value = Wert der Konstante.
String	=	BYTE(33H) StrRef string name exported. string = Text dieser Stringkonstante.
Type	=	BYTE(34H) StrRef ModRef name exported.

Procedure	=	BYTE(35H) ProcNo level address size registers name exported. ProcNo = Nummer der Prozedur. level = Schachtelungstiefe der Prozedur (0=global). address = Adresse der Prozedur. size = Grösse der lokalen Variablen. registers = BITSET, der alle als Parameter verwendeten Register enthält.
Function	=	BYTE(36H) ResRef ProcNo level address size registers name exported. ResRef = Resultattyp der Funktion, sonst wie Procedure.
Module	=	BYTE(37H) ModNo name exported. ModNo = Nummer dieses lokalen Moduls.
Code	=	BYTE(38H) cnum name exported. cnum = Offset dieser Procedure (CODE).
CodeFunc	=	BYTE(39H) ResRef cnum name exported. ResRef = Resultattyp der Funktion.
key	=	"ARRAY [0..2] OF INTEGER".
name	=	string.
string	=	length {char}.

Alle oben nicht erwähnten Symbole sind Zahlen, die nach einem speziellen Verfahren kodiert werden. Dabei richtet sich die Zahl der verwendeten Byte nicht nach der maximalen Grösse des Wertes, sondern nach dem effektiven Wert.

Zahlen im Bereich -32 bis 31 werden in einem Byte kodiert. Dazu werden nur 6 Bit benötigt. Die höchstwertigen zwei Bit werden nicht benötigt. Sie erhalten den Wert 00 um anzugeben, dass die Zahl in einem Byte kodiert wurde.

Zahlen im Bereich -16384..-257 und 256.. 16383 werden in zwei Byte kodiert. Dazu werden nur 14 Bit benötigt, und die zwei höchstwertigen Bit geben wiederum das Format an. Sie erhalten den Wert 01 um anzugeben, dass es sich um eine Zahl handelt, die in zwei Byte abgelegt ist.

Zahlen, die 17 bis 22 Bit zu ihrer Darstellung benötigen, werden in drei Byte abgelegt, wobei die höchstwertigen Bit den Wert 10 (Dualdarstellung) haben.

Für alle anderen Zahlen wird im höchstwertigen Byte nur die Länge (in Byte) der Zahl abgelegt. Die Länge wird in den 6 niederwertigsten Bit des höchstwertigen Byte abgelegt. Die zwei höchstwertigen Bit enthalten den Wert 11, um diese Codierung zu identifizieren. Auf dieses Byte folgen soviele Byte, wie im ersten Byte angegeben wird.

Einige Beispiele:

Zahl	Codierung (Hexadezimal und Binär)
5	05 0000 0101
-5	3B 0011 1011
40	C128 1100 0001 0010 1000
300	412C 0100 0001 0010 1100
-300	7ED4 0111 1110 1101 0100
65236	C20ED4 1100 0010 0000 1110 1101 0100

PCPos stellt davon eine Ausnahme von dieser Kodierung dar. Zahlen die sich im Bereich -32..31 befinden werden nicht in einem einzigen Byte kodiert, sondern in 2 Byte, z.B 3 als C103. Auf diese Weise ist es möglich einen PC-Block von den anderen Elementen der ObjList zu unterscheiden, ohne dass ein zusätzliches Byte vor dem PC-Block stehen muss.

Alle Namen, die mit "Ref" enden, stellen Referenzen auf Strukturen dar. Die Strukturen werden durchnumeriert, wobei die Werte 1 bis 22 vordefinierten Typen entsprechen. Alle in der Symboldatei selbst auftretenden Strukturdefinitionen werden vom Wert 32 an aufsteigend durchnumeriert. Falls eine Referenz auf diese Struktur benötigt wird, ist in der Symboldatei die Nummer der Struktur abgelegt. Beim Einlesen der Datei wird daraus ein Zeiger auf die entsprechende Struktur erzeugt. Die vordefinierten Strukturen sind:

```
1       Undefiniert         12 ADDRESS
2       BOOLEAN             13 BYTE
3       CHAR                14 WORD
4       INTEGER             15 LONGCARD
5       CARDINAL            16 UniversalInteger
6       LONGINT             17 UniversalReal
7       REAL                18 FFP
8       LONGREAL            19 LONGSET
9       BITSET              20 ShortCard
10      PROC                21 ShortInt
11      String              22 BPTR
```

Die Strukturen 1, 11, 16, 17, 20 und 21 werden nur intern gebraucht und stellen keine vordefinierten Typen dar.

Die Symboldatei stellt die Symboltabelle des Compilers dar. In ihr sind alle deklarierten Objekte (Variablen, Typen, Prozeduren etc.) und deren Strukturen (ARRAY, RECORD etc.) abgelegt. Die gewählte Darstellung erlaubt eine effiziente Rekoinstruktion der Symboltabelle beim Einlesen der Symboldatei.

Beispiel

Das folgende Modul wird mit dem Compiler übersetzt und erzeugt eine Symboldatei.

```
DEFINITION MODULE x;

TYPE
 Enum=(rot,gelb,gruen);
 Set=SET OF [0..30];
 Array=ARRAY Enum OF Set;

END x.
```

```
0000: 00000002  Reffile
0004: 0000  ModAnchor
0005:  0F7A 0448 0BB5 Key
000B:  02 78 Name "x"
000D:  01 ModMode
```

```
000E: 10  enum
000F:  01 Size
0010:  03 NoConst
0011: 32  Const
0012:  C120 StrRef
0014:  00 ModRef
0015:  00 value
0016:  04 72 6F 74 Name "rot"
001A:  00 exported
001B: 32  Const
001C:  C120 StrRef
001E:  00 ModRef
001F:  01 value
0020:  05 67 65 6C 62 Name "gelb"
0025:  00 exported
0026: 32  Const
0027:  C120 StrRef
0029:  00 ModRef
002A:  02 value
002B:  06 67 72 75 65 6E Name "gruen"
0031:  00 exported
0032: 34  Type
0033:  C120 StrRef
0035:  00 ModRef
0036:  05 45 6E 75 6D Name "Enum"
003B:  00 exported
003C: 11  range
003D:  01 Size
003E:  10 BaseRef
003F:  00 min
0040:  1E max
0041:  00 sign
0042: 13  set
0043:  04 Size
0044:  C121 BaseRef
0046: 34  Type
0047:  C122 StrRef
0049:  00 ModRef
004A:  04 53 65 74 Name "Set"
004E:  00 exported
004F: 11  range
0050:  00 Size
0051:  C120 BaseRef
0053:  00 min
0054:  02 max
0055:  00 sign
0056: 16  Array
0057:  0C Size
```

```
0058:  C122 ElemRef
005A:  C123 IndexRef
005C: 34  Type
005D:  C124 StrRef
005F:  00 ModRef
0060:  06 41 72 72 61 79 Name "Array"
0066:  00 exported
0067: 0001  ModTag
0068:  00 ModNo
0069: 0003  RefTag
006A:  3E adr
006B:  00 pno
```

0000	Identifikation als Symboldatei
0004	Dieses Modul hat den Namen x und ist ein normales Definitionsmodul.
000E	Diese Struktur bezeichnet einen Aufzählungstyp. Als erste Struktur in der Datei erhält sie die Nummer 32. Der Aufzählungstyp enthält drei Elemente und benötigt 1 Byte Speicherplatz.
0011	Diese Konstante hat als StrRef den Wert C120. Dies ist die kodierte Darstellung von 32. Dies bedeutet, dass diese Konstante zum Typ 32 gehört, dem soeben gefundenen Aufzählungstyp. Die Konstante ist in diesem Modul vereinbart worden, hat den Namen "rot" und den Wert 0. exported wird nicht verwendet, deshalb steht dort der Wert 0.
001B,0026	Konstanten "gelb" und "gruen" des Aufzählungtyps.
0032	Das Objekt des Aufzählungtyps, er heisst Enum und ist in diesem Modul deklariert worden.

003C Diese Struktur bezeichnet einen Unterbereichstyp. Als zweite Struktur in der Datei erhält sie die Nummer 33. Dieser Typ braucht 1 Byte, hat als Basistyp UniversalInteger, als untere Grenze den Wert 0 und als obere Grenze den Wert 30. Dieser Typ hat kein Vorzeichen, er ist also mit Variablen vom Typ CARDINAL und LONGCARD kompatibel, nicht aber mit Variablen vom Typ INTEGER oder LONGINT.

0042 Diese Struktur (Nummer 34) bezeichnet einen Mengentyp der als Elementtyp den vorher deklarierten Unterbereichstyp besitzt. 4 Byte sind nötig für dessen Darstellung.

0046 Das Objekt des Mengentyps. Der Typ ist in diesem Modul deklariert worden und hat den Namen "Set".

004F Struktur 35: Unterbereichstyp mit Basistyp 32 (der Aufzählungstyp), unterer Grenze 0 und oberer Grenze 2.

0056 Struktur 36: Arraytyp, Grösse 12 Byte. Hat Struktur 34 (Menge) als Elementtyp und Struktur 35 (Unterbereich des Aufzählungtyps) als Indextyp.

005C Objekt des Arraytyps: in diesem Modul deklariert, hat Namen "Array".

0067 Das ModTag markiert das Ende des Moduls 0.

0069 Das RefTag markiert das Ende des Definitionsmoduls. Da in diesem Modul keine Variablen deklariert wurden, ist die Adresse, von der an Variablen im Implementationsmodul alloziert werden -2. Die Zahl der deklarierten Prozeduren ist 0.

Ein Typ-Objekt existiert immer dann, wenn einer Struktur in einer Typendefinition ein Name gegeben wurde. Der Unterbereichstyp der in der Mengendefinition deklariert wurde, hat deshalb kein Typ-Objekt.

Der Unterbereichstyp mit der Nummer 35 wurde vom Compiler erzeugt, weil der Indextyp intern immer ein Unterbereichstyp sein muss.

Fehlerdatei

Die Fehlerdatei wird vom Compiler beim Übersetzen eines fehlerhaften Programms erzeugt. Sie enthält die Fehlermeldungen in kodierter Form. m2emacs und m2error sind in der Lage diese Fehlermeldungen mit Hilfe der Datei M2:Fehler-Meldungen in deutschen Text umzuwandeln. Das Format der Fehlerdatei ist:

ErrorFile	=	ERRFILE {Error}.
Error	=	ErrorPos {ErrorPart}.
ErrorPos	=	ERR SourcePos.
ErrorPart	=	ErrorNum | String.
ErrorNum	=	"INTEGER". Nur Zahlen im Bereich 0..32767 erlaubt !
String	=	STR {char}.
char	=	"CHAR". Nur Codes kleiner gleich 177C erlaubt !
SourcePos	=	"LONGINT".
ERRFILE	=	"LONGINT(3)".
ERR	=	"LONGINT(0C1457272H)".
STR	=	"CHAR(0C2H)".

"String" muss mit einem oder zwei 0C-Zeichen abgeschlossen sein. Ein zweites 0C-Zeichen ist manchmal nötig um sicherzustellen, dass das nächste Element der Fehlerdatei auf eine gerade Adresse zu liegen kommt. SourcePos gibt die Position des Fehlers im Quelltext an. ErrorNum bezieht sich auf Meldungen, die in der Datei M2:Fehler-Meldungen gespeichert sind. Diese Datei enthält alle Fehlermeldungen, bzw. Textbausteine die für Fehlermeldungen benötigt werden zusammen mit der Zahl mit der sie identifiziert werden. Die Struktur der Datei ist:

MessageFile	=	{Message} End.
Message	=	Next Number String.
ErrorPos	=	ERR SourcePos.
ErrorPart	=	ErrorNum | String.
Next	=	"LONGINT". Adresse der nächsten Fehlermeldung relativ zum Anfang der Datei.
Number	=	"INTEGER". Fehlernummer.
String	=	{char}. Fehlertext. Muss mit einem 0C abgeschlossen sein. Ein weiteres Zeichen wird, wo nötig, angefügt um sicherzustellen dass die nächste Meldung wieder auf eine gerade Adresse zu liegen kommt.
char	=	"CHAR".
End	=	"LONGINT(0)" "INTEGER(0)".

Laufzeitumgebung von M2Amiga

Ein Modula-2-Programm besteht aus einem oder mehreren Moduln. Zu jedem Modul werden zwei Speicherbereiche, ein Datenbereich und ein Codebereich, angelegt. Im Datenbereich werden die globalen Daten eines Moduls abgelegt. Der Codebereich enthält neben dem eigentlichen Code auch alle konstanten Zeichenketten, eine Tabelle mit Sprungbefehlen zu den exportierten Prozeduren, eine Tabelle mit den Adressen aller importierten Module und einen Zeiger auf den Datenbereich (Fig. 1, niedrige Adressen oben, höhere Adressen unten)

Der Modulzeiger zeiget auf das Ende der Prozedurtabelle. Dies erlaubt Moduln gleich wie Amiga-Libraries anzuspringen, nämlich indem das Register `A6` mit der Adresse des Moduls beziehungsweise der Library geladen wird und dann mit einem `JSR offset(A6)` in die Prozedurtabelle gesprungen wird.

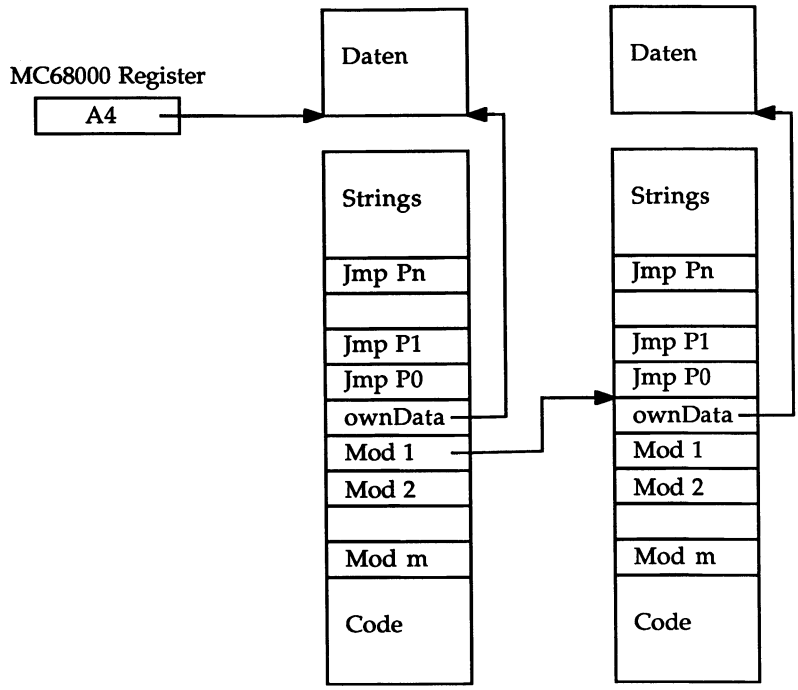


Fig. 1

Der Zeiger auf den eigenen Datenbereich zeigt immer auf dessen Ende. Während des Programmablaufs zeigt das A4-Register immer auf die globalen Daten des eigenen Moduls (static base). An der Stelle -2 (A4) befindet sich das Flag, d.h ein Wort, das den Wert 0 hat, wenn das Modul noch nicht initialisiert ist, und das auf 1 gesetzt wird wenn das Modul initialisiert wurde. Die globalen Daten werden in negativer Richtung alloziert. Dabei werden alle Variablen, die grösser als ein Byte sind immer auf eine gerade Adresse gelegt.

Beispiel:
```
VAR
 i:INTEGER;                    -4
 c:CHAR;                       -5
 a:ARRAY [0..5] OF CHAR;      -12
```

Das A5-Register zeigt auf den Prozedurdeskriptor (mark pointer). Der Prozedurdeskriptor umfasst 12 Byte. Im Falle einer globalen Prozedur enthält sie neben der Rücksprungadresse einen Zeiger auf den Prozedurdeskriptor der aufrufenden Prozedur (dynamic link) und den alten Zustand des A4-Registers. Bei einer lokalen Prozedur ist es nicht nötig, das Register A4 zu retten, da kein Modulwechsel stattfindet. Stattdessen wird aber ein Zeiger auf den aktuellste Prozedurdeskriptor der umschliessenden Prozedur benötigt (static link). Die beiden Arten von Prozedurdeskriptoren sind in Fig. 2 dargestellt.

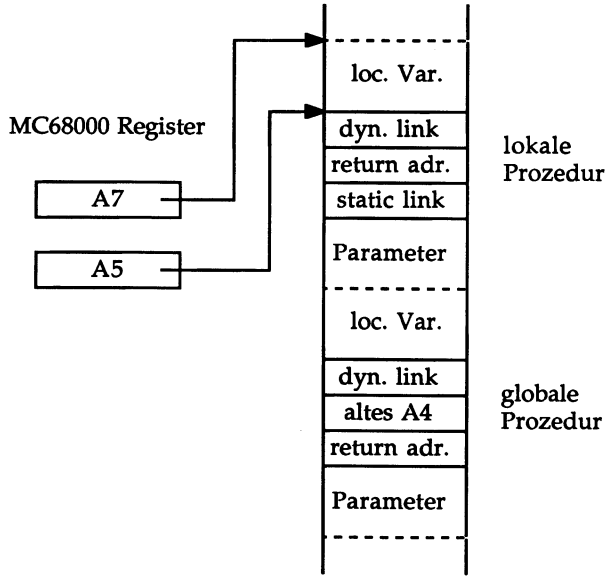


Fig. 2

Vom Prozedurdeskriptor aus in negativer Richtung werden die lokalen Variablen angelegt. Mit positivem Offset zum A5-Register erreicht man die Parameter der Prozedur. Sowohl für die Parameter als auch für die lokalen Variablen gilt dieselbe Art der Adressenvergabe wie für die globalen Variablen.

Parameter können in Modula-2 als Wert- oder als Variablenparameter übergeben werden. Bei Wertparametern wird immer der Wert der

Variable auf den Stack kopiert. So sollte man daran denken, einen genügend grossen Stack bereitzustellen, falls eine Prozedur beispielsweise ein grosses Feld als Werteparameter besitzt. Bei Variablenparameter wird nur die Adresse der Variablen auf den Stack kopiert. Die Prozedur benutzt dann diese Adresse, um auf den Inhalt der übergebenen Variablen zugreifen zu können.

Eine Ausnahme von dieser Regel tritt bei den offenen Feldparametern ein. Da nicht von vornherein bekannt ist, wie gross das Array beim Aufruf ist (selbst bei Werteparametern) kann nicht der Inhalt der Variablen auf den Stack kopiert werden, da die Prozedur sonst nicht wüsste wie sie die nachfolgenden Parameter adressieren sollte. Es wird in diesem Fall ein 8 Byte grosser Deskriptor auf den Stack kopiert. Dieser enthält den Wert von HIGH und die Adresse der Variablen. Die Prozedur kopiert, wenn es sich um einen Werteparameter handelt, die Variable, an die lokalen Variablen anschliessend, auf den Stack.

Kontrollcode

Der Quelltext eines Modula-2-Programms wird vom Compiler bereits auf grösstmögliche Fehlerfreiheit kontrolliert. Der Compiler prüft, ob die Syntax des Programms korrekt ist. Einige Fehler sind jedoch erst zur Laufzeit des Programms feststellbar.

Dazu wird vom Compiler Kontrollcode erzeugt, der je nach Situation durchlaufen wird. Signalisiert dieser Kontrollcode eine abnormale Situation so resultiert ein Laufzeitfehler. Dies bringt dem Progammierer zwei entscheidende Vorteile:

- Fehler in der Programmierung können genauer lokalisiert werden und die Fehlersuche wird kürzer und effektiver.

- Das Programm wird im Fehlerfall sauber abgebrochen, die anderen Prozesse laufen ungestört weiter. Das hat zur Konsequenz, dass der ungeliebte "Guru" nicht erscheint.

Diesen Vorteilen stehen aber auch Nachteile gegenüber:

- Der Kontrollcode vergrössert die Objektdatei.

- Das Programm wird langsamer.

Anhand der folgenden drei Beispielprozeduren soll die Auswirkung des Kontrollcodes auf Grösse und Geschwindigkeit des erzeugten Code gezeigt werden. Die *kursiv* gedruckten Zeilen heben den Kontrollcode hervor:

```
CONST
 Range=2000000;

PROCEDURE NoRangeCheck;
VAR
 i:LONGINT;
 j:INTEGER;
 a:ARRAY[0..7] OF INTEGER;
BEGIN
 j:=5;
 FOR i:=0 TO Range DO
  (*$R-*)
  a[j]:=2;

01AA:3C2D FFFA           MOVE.W  -6(A5), D6
01AE:E346                ASL.W   #1, D6
01B0:47ED FFEA           LEA     -22(A5), A3
01B4:37BC 00026000       MOVE.W  #2, 0(A3, D6.W)

  (*$R=*)
 END;
END NoRangeCheck;
```

```
PROCEDURE RangeCheck;
VAR
 i:LONGINT;
 j:INTEGER;
 a:ARRAY[0..7] OF INTEGER;
BEGIN
 j:=5;
 FOR i:=0 TO Range DO
  (*$R+*)
  a[j]:=2;

016A:3C2D FFFA           MOVE.W  -6(A5), D6
016E:4DBC 0007           CHK     #7, D6
0172:E346                ASL.W   #1, D6
0174:47ED FFEA           LEA     -22(A5), A3
0178:37BC 00026000       MOVE.W  #2, 0(A3, D6.W)

  (*$R=*)
 END;
END RangeCheck;


CONST
 Overflow=2000000;

PROCEDURE NoOverflowCheck;
VAR
 i:LONGINT;
 n:LONGINT;
BEGIN
 n:=0;
 FOR i:=0 TO Overflow DO
   (*$V-*)
  n:=n+1;

0134:2C2D FFF8           MOVE.L  -8(A5), D6
0138:5286                ADDQ.L  #1, D6
013A:2B46 FFF8           MOVE.L  D6, -8(A5)

   (*$V=*)
 END;
END NoOverflowCheck;
```

```
PROCEDURE OverflowCheck;
VAR
 i:LONGINT;
 n:LONGINT;
BEGIN
 n:=0;
 FOR i:=0 TO Overflow DO
   (*$V+*)
  n:=n+1;

00FE:2C2D FFF8            MOVE.L  -8(A5), D6
0102:5286                 ADDQ.L  #1, D6
0104:4E76                 TRAPV
0106:2B46 FFF8            MOVE.L  D6, -8(A5)

   (*$V-*)
 END;
END OverflowCheck;

CONST
Stacks=1000000;


(*$S-*)
PROCEDURE NoStackCheck2;
VAR
 i:ARRAY[0..99] OF LONGINT;
BEGIN

0218:2F0C                 MOVE.L  A4, -(A7)
021A:287A FEA8            MOVE.L  -344(PC)<000000C4>, A4
021E:4E55 FE70            LINK    A5, #-400

END NoStackCheck2;
(*$S=*)
```

```
PROCEDURE NoStackCheck;
VAR
 i:LONGINT;
BEGIN
 FOR i:=0 TO Stacks DO
   (*$S-*)
  NoStackCheck2;

0242:6100 FFD4            BSR     00000218  [-44]

   (*$S=*)
 END;
END NoStackCheck;


(*$S+*)
PROCEDURE StackCheck2;
VAR
 i:ARRAY[0..99] OF LONGINT;
BEGIN

01C6:203C FFFFFE70        MOVE.L  #-400, D0
01CC:2C7A FEFA            MOVE.L  -262(PC)<000000C8>, A6
01D0:4EAE FFEE            JSR     -18(A6)
01D4:2F0C                 MOVE.L  A4, -(A7)
01D6:287A FEEC            MOVE.L  -276(PC)<000000C4>, A4
01DA:4E55 FE70            LINK    A5, #-400

END StackCheck2;
(*$S=*)


PROCEDURE StackCheck;
VAR
 i:LONGINT;
BEGIN
 FOR i:=0 TO Stacks DO
  (*$S+*)
  StackCheck2;

01FE:70F4                 MOVEQ   #-12, D0
0200:2C7A FEC6            MOVE.L  -314(PC)<000000C8>, A6
0204:4EAE FFEE            JSR     -18(A6)
0208:6100 FFBC            BSR     000001C6  [-68]

  (*$S=*)
 END;
END StackCheck;
```

Bereichs- und Überlaufprüfung benötigen wenig zusätzlichen Code. In diese Prozeduren wurde nur jeweils eine Anweisung eingefügt. Anders verhält es sich mit der Stapelüberlaufprüfung. Beim Aufruf der Prozedur müssen 3 zusätzliche Befehle eingefügt werden (10 Byte). Dasselbe geschieht beim Eingangscode der aufgerufenen Prozedur. Der eingefügte Code ruft eine Prozedur im Modul Arts auf, welche überprüft, ob die benötigte Anzahl von Byte auf dem Stapel noch verfügbar ist. Diese Überprüfungen kosten nicht nur Speicherplatz, sondern auch Ausführungszeit:

Überprüfung	ausgeschaltet	eingeschaltet	Zunahme
Bereich	33.9 s	35.5 s	5 %
Überlauf	36.2 s	40.5 s	12 %
Stapelüberlauf	26.9 s	81 s	201 %

Währenddem die Bereichs- und Überlaufprüfungen kaum zusätzlich Zeit beanspruchen, braucht der Aufruf einer leeren Prozedur 3 mal mehr Zeit wenn die Überprüfung eingeschaltet ist. Sollte also ein Programmteil nicht schnell genug laufen, empfiehlt es sich als erstes die Stapelüberlaufprüfung auszuschalten, weil man damit das Programm am ehesten beschleunigen kann.

11 Programmierhinweise

Übersetzung von C-Programmen 3
Aufzählungs- und Mengentypen 3
VAR-Parameter 4
Record-Varianten 6
Funktionsresultate 7

M2Amiga Besonderheiten 8
Zeichenketten 8
Terminate statt Exit 9
Verwendung von Abschlussprozeduren und Assert 9

In diesem Kapitel soll auf einige Besonderheiten bei der Programmierung mit M2Amiga hingewiesen werden. Das Betriebsystem des Amigas wurde in den Programmiersprachen C, Assembler und BCPL implementiert. Eigenheiten dieser Sprachen haben an verschiedenen Stellen Spuren hinterlassen. Um Modula-2 optimal in diese Umgebung einzufügen ist es deshalb nötig deren Spezialitäten zu kennen.

Übersetzung von C-Programmen

Sollen auf dem Amiga Betriebssystem-Schnittstellen verwendet werden, so ist meist die Kenntnis der Programmiersprache C vonnöten. Auch scheinen die Deklarationen von Amiga-Modula-2 nicht immer einleuchtend. Dieser Abschnitt soll einige Unterschiede darstellen.

Aufzählungs- und Mengentypen

In C findet man oft Deklarationen wie:

```
SERB_parityOn=0;
SERB_parityOdd=1;                                          <A>
SERB_sevenWire=2;
SERB_queuedBrk=3;


SERF_parityOn =0x0001; (oder 1<<SERB_parityOn)
SERF_parityOdd=0x0002; (oder 1<<SERB_parityOdd)            <B>
SERF_sevenWire=0x0004; (oder 1<<SERB_sevenWire)
SERF_queuedBrk=0x0008; (oder 1<<SERB_queuedBrk)
```

Im Programm steht dann:

```
flags=SERF_parityOn|SERF_queuedBrk;                        <C>
```

Dies heisst in Modula-2 viel eleganter:

```
SerFlags=(parityOn,parityOdd,sevenWire,queuedBrk);         <A>

SerFlagSet=SET OF SerFlags;                                <B>
```

```
flags:=SerFlagSet{parityOn,queuedBrk};                    <C>
```

Ein Aufzählungstyp in Modula-2 ist gleichbedeutend mit einer Deklaration von Konstanten, die von 0 an durchnumeriert werden. Deshalb kann die C-Deklaration <A> in einen Aufzählungstyp umgewandelt werden.

Eine Menge wird intern so dargestellt, dass jedes Element ein Bit beansprucht, und zwar so, dass das erste Element die Bitposition 0, das zweite Element die Bitposition 1 usw.. annimmt. Deshalb kann die C-Deklaration <B> in einen Mengentyp umgewandelt werden.

Die Anweisung <C> ergibt sich als logische Konsequenz der Definition.

Nebst der überschaubareren Schreibweise ermöglicht diese Übersetzung dem Compiler eine strengere Typenkontrolle. Die Elemente der Aufzählungs- und der Mengentypen sind an ihren Typ gebunden. So sind "ScreenFlags" und "WindowFlags" nur an Variablen ihres Typs zuweisbar. Eine Verletzung dieser Regel wird schon vom Compiler beanstandet und nicht erst zur Laufzeit durch Guru Messages gemeldet.

VAR-Parameter

Ein weiterer Vorteil von Modula-2 gegenüber der Sprache C sind die VAR-Parameter. C kennt ausschliesslich Werteparameter, d.h. die Prozedur kann den ihr übergebenen Wert nur lokal verändern. Soll der Inhalt eines Parameters in der Prozedur bleibend verändert werden, dann muss man einige Kunstgriffe anwenden. Was in Modula-2 so einfach aussieht:

```
MODULE m;

PROCEDURE P(i:INTEGER; VAR j:INTEGER);
BEGIN
 j:=2*i;
END P;

VAR
 k:INTEGER;
BEGIN
 P(2,k);
END m.
```

wird in C folgendermassen geschrieben:

```
void p(i,j)

int i;
int *j; /* j wird als Zeiger auf int deklariert */

{
 (*j)=2*i;
}

main()
{
 int k;

 p(2,&k); /* Für j muss Adresse übergeben werden */
}
```

Das bedeutet, dass man beim Zurückübersetzen nach Modula-2 aufpassen muss, nicht wörtlich zu übersetzen, sonst erhält man ein Programm, dass zwar richtig ist und funktioniert, aber seine "C-Vergangenheit" nicht verbergen kann:

```
MODULE m;

FROM SYSTEM IMPORT ADR;

TYPE
 IntegerPtr=POINTER TO INTEGER;

PROCEDURE P(I:INTEGER; j:IntegerPtr);
BEGIN
 j^:=2*i;
END P;

VAR
 k:INTEGER;
BEGIN
 P(2,ADR(k));
END m.
```

Beispiele für eine elegante Übersetzung in Modula-2 findet man im Modul Graphics. Die Prozedur `InitBitmap` erwartet als ersten Parameter die Adresse einer `BitMap`. In Modula-2 wurde das als ein VAR-Parameter des Typs `BitMap` deklariert. Die Übergabe mit Hilfe von VAR-Parametern wurde überall dort so übersetzt, wo es nicht unbedingt nötig ist, einen Zeiger zu übergeben.

Record-Varianten

Union in C und Varianten im RECORD in Modula-2 dienen dem gleichen Zweck, sind aber nicht identisch. In beiden Fällen geht es darum, verschiedene Felder eines RECORDs auf die gleiche Speicherstelle abzubilden, um so zwischen den Typen zu wechseln. Wesentlicher Unterschied: In C umfasst ein union die verschiedenen Darstellungen eines einzelnen Feldes, während in Modula-2 mehrere Felder angegeben werden können. Bsp:

```
struct example {
 LONG first;
 union {
  WORD a;
  BYTE b;
  char c;
 } second;
};
```

ist in Modula-2

```
example=RECORD
 first:LONGINT;
 CASE :INTEGER OF
 | 0: a:CARDINAL;
 | 1: b:BYTE;
 | 2: c:CHAR;
 END;
END;
```

Ein weiterer Unterschied ist, dass man in C einen Namen für den union und einzelne Namen für die verschiedenen Varianten angibt. In Modula-2 ist dies nicht der Fall. Bsp, Zugriff auf das Feld b:

C:	example.second.b
Modula-2:	example.b

Dafür können in Modula-2 die Varianten nur auf RECORDs angewendet werden, während in C auch einfache Variablen als union deklariert werden können.

Funktionsresultate

Die Prozedur `DateStamp` aus Dos gibt den Pointer, den man als Parameter übergibt, als Funktionsresultat wieder zurück. Dies erlaubt es, in C zu schreiben:

```
sekunden=DateStamp(v)->secs;
```

Dies ist in Modula-2 nicht möglich, der entsprechende Befehl

```
sekunden:=DateStamp(v)^.secs;
```

ist in Modula-2 ungültig. In Modula-2 programmiert man dies üblicherweise als:

```
DateStamp(v);
sekunden:=v^.secs;
```

Die Prozedur `DateStamp` hat keinen Resultatswert mehr.

Aus ähnlichen Überlegungen heraus sind auch die Prozeduren `OpenDevice` und `DoIO` aus `Exec` als reine Prozeduren formuliert. Der zurückgegebene Resultatswert ist nämlich nichts anderes als der Fehlerwert im `error`-Feld der `IORequest`-Struktur. In Modula-2 wird man also statt

```
if (DoIO(request)=0) { .... };
```

schreiben

```
DoIO(request); IF request.error=0 THEN ... END;
```

M2Amiga Besonderheiten

Einige Besonderheiten ergeben sich daraus, dass gewisse Konventionen die in Modula-2 gelten, nicht identisch sind mit den entsprechenden Konventionen des Amiga.

Zeichenketten

Zeichenketten werden in Modula-2 in Variablen des Typs `ARRAY [0..n] OF CHAR` abgelegt. Die darin abgelegte Zeichenkette darf 0 bis n+1 Zeichen umfassen. Ist sie kürzer als n+1, dann wird sie mit einem 0C Zeichen abgeschlossen. Ist die Zeichenkette jedoch genau n+1 Zeichen lang, dann ist sie *nicht* mit einem 0C Zeichen abgeschlossen. Die Amiga Prozeduren erwarten jedoch, dass *jede* Zeichenkette mit einem 0C Zeichen abgeschlossen ist. Man ist also dafür verantwortlich, dass man seine Variablen lang genug deklariert.

Besonders vorsichtig muss man bei der Übergabe von konstanten Zeichenketten an einen offenen Feldparametern sein. In diesem Fall werden nämlich nur die signifikanten Zeichen übergeben. Das

abschliessende 0C fehlt.. Einer Modula-Prozedur, wie z.B. Terminal.WriteString, macht dies nichts aus, denn sie wird mithilfe der Modula-2 Funktion HIGH die Länge der Zeichenkette herausfinden können.

Anders verhält es sich aber mit den Prozeduren der Schnittstellen-Module, wie z.B Dos.Open. Dieser Prozedur muss der Name der zu öffnenden Datei übergeben werden können. In M2Amiga geschieht dies, indem der Parameter nicht eine Zeichenkette, sondern eine Adresse erwartet, also:

```
f:=Dos.Open(ADR("s:Startup-Sequence"),oldFile);
```

Dies funktioniert, weil ADR die Adresse auf die Zeichenkette übergibt, die sich im Konstantenteil des Codebereichs befindet. Im Konstantenbereich ist jede Zeichenkette garantiert mit einem 0C abgeschlossen.

Terminate statt Exit

Dos stellt zur Beendigung eines Programmes die Prozedur `Exit` zur Verfügung. Diese sollte man in Modula-2 Programmen nicht verwenden. Exit würde nämlich das Programm unmittelbar beenden, ohne dem Ausgangscode von Arts die Möglichkeit zu geben die Abschlussprozeduren aufzurufen. Das autmatische Freigeben der Resourcen (z.B. Speicher der mit Heap alloziert wurde) würde dann nicht ausgeführt!

Falls ein Abbruch des Programms benötigt wird, sollte die Prozedur Terminate aus dem Modul Arts benutzt werden. Eine Mdula-2 gerechte Exit Prozedur sieht dann so aus:

```
PROCEDURE Exit(rc:LONGINT);
BEGIN
 returnVal:=rc;
 Terminate(CurrentLevel());
END Exit;
```

Verwendung von Abschlussprozeduren und Assert

Die Abschlussprozeduren und die Prozedur Assert stellen ein bequemes Mittel dar, um Programme sicher abzubrechen, falls ein Fehler aufgetreten ist, der eine weitere Ausführung des Programms nicht erlaubt. Folgendes Beispielprogramm soll das illustrieren:

```
MODULE m;
...
VAR
 screen:ScreenPtr;
 window;WindowPtr;

PROCEDURE Cleanup;
BEGIN
 IF window#NIL THEN
  CloseWindow(window); window:=NIL;
 END;
 IF screen#NIL THEN
  CloseScreen(screen); screen:=NIL;
 END;
END Cleanup;

PROCEDURE Init;
CONST
 screenError="Screen nicht geöffnet";
 windowError="Fenster nicht geöffnet";
VAR
 newScr:NewScreen;
 newWin:NewWindow;
BEGIN
 screen:=NIL; window;=NIL;
 TermProcedure(Cleanup);
  ... initialisiere newScr ...
 screen:=OpenScreen(newScr);
 Assert(screen#NIL,ADR(screenError));
  ... initialisiere newWin ...
 window:=OpenWindow(newWin);
 Assert(window#NIL,ADR(windowError));
END Init;

BEGIN
 Init;
  ... von hier an ist garantiert Alles offen.
END m.
```

Die Prozedur `Cleanup` stellt die Abschlussprozedur dar. Sie geht davon aus, dass in `window` bzw. `screen` nur genau dann ein Wert ungleich NIL gespeichert ist, wenn das Fenster bzw. der Screen geöffnet ist. Ist dies bei einer der beiden Variablen der Fall, dann wird das Fenster bzw. der Screen geschlossen.

`Init` initialisiert als erstes die Variablen `window` und `screen` mit dem Wert NIL. Nachdem diese Anweisungen ausgeführt sind soll sind die obengenannten Voraussetzungen für das korrekte Funktionieren von `Cleanup` erfüllt. Die `Cleanup` Prozedur kann nun als Abschlussprozedur mit TermProcedure(Cleanup) installiert werden.

Nachdem dieCleanup Prozedur installiert ist, wird versucht den Screen zu öffnen. `Assert` überprüft, ob dies gelungen ist. Sollte `screen` nach dem `OpenScreen` Befehl immer noch den Wert NIL haben, dann ist etwas schiefgelaufen. `Assert` würden dann das Programm mit dem Requester "Screen nicht geöffnet" beenden. Dasselbe geschieht danach mit dem Fenster.

Man beachte dabei das korrekte Zusammenspiel zwischen Initialisierung- und Abschlussprozedur. Nehmen wir an, das Öffnen des Screen gelingt, nicht aber das Öffnen des Fensters. Beim Aufruf des zweiten `Assert` Befehls wird das Programm abgebrochen, wobei `screen` einen Wert ungleich NIL hat (Screen offen), und `window` den Wert NIL hat. Die Abschlussprozedur `Cleanup`, die nun aufgerufen wird, schliesst den Screen, wird aber nicht versuchen das Fenster zu schliessen (was unweigerlich zu einem Absturz führen würde), da `window` den Wert NIL hat.

12 Bibliotheken

Arguments 3

Arts 7

ASCII 11

Assembler 12

Conversions 17

Coroutines 20

FFPConversions 22

FFPInOut 24

FileMessage 25

FileNames 26

FileSystem 28

Erkennen des Dateiendes bei Leseoperationen 31

Heap 33

InOut 36

LongRealConversions 39

LongRealInOut 41

MathLib0 42

MathLibFFP 44

MathLibLong 46

RandomNumber 47

RealConversions 48

RealInOut 50

Scan 51

Storage 52

Str 53

Strings 56

SYSTEM 58

Terminal 59

Verwendeter Ein- und Ausgabekanal 61

Windows 64

Arguments

Das Modul Arguments erlaubt den Zugriff auf die Programmargumente, die der Benutzer beim Aufstarten des Programms angibt. Unterstützt werden Argumente, die über das CLI auf der Befehlszeile oder über die Workbench durch erweiterte Selektion, übergeben werden.

```
DEFINITION MODULE Arguments;

FROM Dos IMPORT
 FileLockPtr;

VAR
 quoted: BOOLEAN;

PROCEDURE NumArgs(): INTEGER;

PROCEDURE GetArg(arg: INTEGER;
                 VAR argument: ARRAY OF CHAR;
                 VAR len: INTEGER);

PROCEDURE GetLock(arg: INTEGER): FileLockPtr;

END Arguments.
```

Die Prozedur `NumArgs` liefert die Anzahl der Programmargumente. Diese Zahl gibt entweder die Anzahl Argumente der Befehlszeile oder die Anzahl selektierter Workbench-Ikonen an. Wurde kein Argument übergeben, liefert `NumArgs` den Wert 0.

`GetArg` liefert den Wert des gewünschten Arguments, sofern dieses existiert. Ein Aufruf mit Argumentnummer `arg=0` liefert den Namen des aufgestarteten Programms. Bei jedem Aufruf von `GetArg` erhält die Variable `quoted` den Wahrheitswert wahr, falls das Argument doppelte Anführungszeichen enthielt. Dies ermöglicht die Unterscheidung von Schlüsselwörtern und Argumenten mit demselben Namen.

Beispiel:	`dir "all"`	zeigt den Inhalt des Verzeichnis "all"
	`dir all`	ausführliches Verzeichnis-Listing

`GetArg` verbirgt die Unterschiede der Parameterübergabe von CLI und Workbench vor dem Benutzer. Während ein Argument der Befehlszeile aus einer einfachen Zeichenkette besteht, setzt sich ein Workbench-Argument aus zwei Komponenten zusammen, dem Namen des Arguments und einer Referenz auf das Verzeichnis der entsprechenden Datei. Bei Workbench-Argumenten setzt der Aufruf von `GetArg` das aktuelle Verzeichnis des Programms auf das Verzeichnis der Argumentdatei (durch `CurrentDir` [DOS]).

Bei der Interpretation der Befehlszeile werden folgende Regeln angewendet: Leerschläge werden grundsätzlich als Trennung zwischen zwei Argumenten ausgelegt. Durch die Verwendung von doppelten Anführungszeichen (") können aber auch Leerschläge in das Argument eingefügt werden. Alle Zeichen, die zwischen zwei Anführungszeichen stehen, werden unverändert in das Argument übernommen; die Anführungszeichen selbst üben keine Trennfunktionen aus.

Ein leeres Argument kann neuerdings auch ein gültges Resultat von GetArg sein. In AmigaDos 1.3 bezeichnen leere Argumente das aktuelle Verzeichnis.

Beispiel: `copy M2Disk:MeinProjekt/Test.mod ""`

Dies kopiert die Datei "`M2Disk:MeinProjekt/Test.mod`" in das aktuelle Verzeichnis.

Alle Programme im Stil

```
argNr:=1;
LOOP
 GetArg(argNr, argStr, argLen); INC(argNr);
 IF argLen = 0 THEN EXIT END;
 ...
END;
```

sollten zu

```
FOR argNr:=1 TO NumArgs() DO
 GetArg(argNr, argStr, argLen);
```

```
  ...
END;
```

umgeschrieben werden.

Die Funktion `GetLock` gibt immer einen gültigen, "gelockten" `FileLockPtr` zurück, der auf das Verzeichnis des zugehörigen Arguments verweist. Dieser `FileLockPtr` muss mittels `Dos.UnLock()` später wieder freigegeben werden. Dieser wird bei Workbench-Argumenten durch `Dos.DupLock()`, während bei CLI-Argumenten das aktuelle Verzeichnis durch einen Aufruf von `Dos.Lock(ADR(""), Dos,sharedLock)` erzeugt wird.

Beispiel:

```
MODULE args;
(* Programm "args" gibt alle übergebenen
 * Argumente über Terminal aus, unabhängig von
 * der Aufruf-Umgebung
 *)
FROM Arguments IMPORT
 GetArg, NumArgs;
FROM ASCII IMPORT
 ht;
FROM Terminal IMPORT
 Write, WriteLn, WriteString;

VAR
 arg, len: INTEGER;
 argument: ARRAY [0..63] OF CHAR;
BEGIN
 FOR i:=0 TO NumArgs() DO
  GetArg(arg, argument, len);
  Write("-"); Write(ht);
  Write('"'); WriteString(argument);
Write('"');
  WriteLn
 END
END args.
```

Einige Aufrufe des obigen Programms:

```
args Max und Moritz ergibt drei Argumente:
   -       "args"
   -       "Max"
   -       "und"
   -       "Moritz"
args "Max und Moritz" ergibt das Argument:
   -       "args"
   -       "Max und Moritz"
args Ma"x und" Moritz ergibt zwei Argumente:
   -       "args"
   -       "Max und"
   -       "Moritz"
```

Arts

Die Schnittstelle zu Arts (Amiga Runtime System; Amiga Laufzeitsystem) erlaubt den Zugriff auf einige Prozeduren des Amiga Modula-2 Laufzeitsystems.

Warnung:	Verwenden Sie diese Prozeduren nur, wenn sie wissen, was diese bewirken! Jede Fehlmanipulation kann zu unvorhersehbaren Resultaten führen.

```
DEFINITION MODULE Arts;

FROM SYSTEM IMPORT
 ADDRESS,LONGSET;

CONST
 maxModName=32; maxKeys=3;

TYPE
 ErrorType=(trap,exception,system,assertion,
  breakPoint,explicit);
 SysErr=(halt,illCase,fctReturn,stkOvl,illCall);
 ErrorFrame=RECORD
  pc: ADRESS;
  dRegs: ARRAY [0..7] OF LONGINT;
  aRegs: ARRAY [8..15] OF ADDRESS;
  CASE error: ErrorType OF
  | trap: trapNr: INTEGER;
  | exception: exceptionMask: LONGSET
  | system: sysErr: SysErr
  | assertion,breakPoint,explicit:
  END;
  header,body: ADDRESS
 END;
(*
 * Folgende Strukturen sind für den Linker,
 * den Loader und Arts.
 * Nicht für allgemeinen Gebrauch!
 *)
 ModType=(none,mod,lib,noImp);
 ModName=ARRAY [0..maxModName-1] OF CHAR;
 ModKeys=ARRAY [0..maxKeys-1] OF CARDINAL;
 ModuleId=RECORD
  type: INTEGER; (* ORD(ModType) *)
```

```
  name: ModName;
  CASE :ModType OF
  | lib:
   pad: INTEGER;
   libVersion: LONGINT;
  | mod, noImp:
   keys: ModKeys;
  END
 END;
 ModuleInfo=RECORD
  id: ModuleId;
  CASE :ModType OF
  | lib:
   libPtr: ADDRESS;
   fixup: ADDRESS
  | mod:
   modPtr: ADDRESS;
   dataPtr: ADDRESS
  END
 END;
 ModuleInfoPtr=POINTER TO ModuleInfo;

VAR
(*
 * Ändern der folgenden Werte garantiert den
 * Programmabsturz!
 * Ursprüngliche CLI-Argumente und Resultatswert
 * (auf 0 gesetzt)
 *)
 stackSize: LONGINT;
 dosCmdBuf: ADDRESS;    (* {a0} *)
 dosCmdLen: LONGINT;    (* {d0} *)
 returnVal: LONGINT;
(*
 * wbStarted ist wahr, falls von der Workbench gestartet
 *)
 wbStarted: BOOLEAN;
 startupMsg: ADDRESS;

(* Modula-2 Umgebungs-Variablen *)
 stackTop: ADDRESS;
 errorFrame: ErrorFrame;
 moduleInfo: ModuleInfoPtr;    (* {d7} *)

(* Diese Prozedur darf nicht aufgerufen werden!!! *)
PROCEDURE Startup(base,stk: LONGINT): LONGINT;

(* Compiler Unterstützung *)
```

```
PROCEDURE StkChk(need{0}: LONGINT);
PROCEDURE SystemError(err{0}: SysErr);
(*SystemError(halt)=HALT*)
PROCEDURE Mulu32(x{0},y{1}: LONGINT): LONGINT;
PROCEDURE Divu32(x{0},y{1}: LONGINT): LONGINT;
PROCEDURE Muls32(x{0},y{1}: LONGINT): LONGINT;
PROCEDURE Divs32(x{0},y{1}: LONGINT): LONGINT;

(* Laufzeitunterstützung *)
PROCEDURE Assert(condition: BOOLEAN; msg: ADDRESS);
PROCEDURE BreakPoint(msg: ADDRESS);
PROCEDURE Call(p: PROC);
PROCEDURE CurrentLevel(): INTEGER;
PROCEDURE DebugProcedure(debugger: PROC);
PROCEDURE DetectCtrlC(on{0}: BOOLEAN);
PROCEDURE Error(header,body: ADDRESS);
PROCEDURE RemoveDebugProc(debugger: PROC);
PROCEDURE RemoveTermProc(p: PROC);
PROCEDURE Requester(
           header,body,pos,neg: ADDRESS): BOOLEAN;
PROCEDURE Terminate(level: INTEGER);
PROCEDURE TermProcedure(p: PROC);

END Arts.
```

Das Modul Arts hat das Erscheinungsbild eines normalen Bibliotheksmoduls, bestehend aus einer Symbol- und einer Objekt-Datei. Die Aufgaben von Arts sind jedoch weitaus vielfältiger und spezieller. Arts ist das sogenannte Laufzeitsystem, das die Basis jedes Modula-2 Programms bildet. Arts wird von jedem Modul implizit

Die Prozeduren `Assert`, `BreakPoint` und `Error` vereinfachen eine einheitliche Fehlerbehandlung. `Assert` kontrolliert, ob eine Bedingung erfüllt ist. Dabei geht es um Bedingungen, die für das Weiterlaufen des Programms unbedingt erfüllt sein müssen (→ Kapitel Programmierhinweise). `BreakPoint` bedingt einen Programmunterbruch, nach welchem wieder weitergefahren wird. Diese Prozedur kommt vor allem zur Anwendung, wenn auch ein Debugger installiert ist, der dann aktiviert werden kann. So wird der Programmzustand an einigen kritischen Punkten kontrolliert. Die Prozedur `Error` verursacht einen Programmabbruch, der an keine Bedingung geknüpft ist. An die Prozeduren `Assert`, `BreakPoint` und `Error` können auch Fehlermeldungen übergeben werden, die im Requester ausgegeben werden.

Die Funktion `Requester` zeigt einen Requester mit den angegebenen Texten `header` und `body`. Die Felder `pos` und `neg` zeigen auf die Texte, die in die Gadgets geschrieben werden. `pos` kann auch `NIL` sein, in diesem Fall wird der Requester nur ein Gadget haben. Der Rückgabewert von `Requester` gibt an, welches Gadget gewählt wurde. Wurde das Gadget mit dem Text von `pos` ausgewählt, dann gibt Requester den Wert `TRUE` zurück, sonst den Wert `FALSE`.

Die Funktion `DetectCtrlC` erlaubt das Abschalten der CTRL-C behandlung. Normalerweise kann der Benutzer durch drücken der CTRL-C Taste oder mithilfe des BREAK-Befehl [DOS] das Programm abbrechen. Dies ist nicht immer erwünscht, z.B. sollte ein Terminalprogramm CTRL-C als normale Eingabe akzeptieren. Ein Aufruf von `DetectCtrlC` mit dem Parameter `on=FALSE` schaltet die CTRL-C Behandlung in Arts ab. Ein Aufruf von `DetectCtrlC` mit dem Parameter `on=TRUE` schaltet sie wieder ein.

ASCII

Das Modul `ASCII` definiert Namen für die ASCII[1]-Steuerzeichen.

```
(* $M- *)
DEFINITION MODULE ASCII;

CONST
 nul = 00C;  soh = 01C;  stx = 02C;  etx = 03C;
 eot = 04C;  enq = 05C;  ack = 06C;  bel = 07C;
 bs  = 10C;  ht  = 11C;  lf  = 12C;  vt  = 13C;
 ff  = 14C;  cr  = 15C;  so  = 16C;  si  = 17C;
 dle = 20C;  dc1 = 21C;  dc2 = 22C;  dc3 = 23C;
 dc4 = 24C;  nak = 25C;  syn = 26C;  etb = 27C;
 can = 30C;  em  = 31C;  sub = 32C;  esc = 33C;
 fs  = 34C;  gs  = 35C;  rs  = 36C;  us  = 37C;
 sp  = ' ';
 del = 177C;
 eol = lf;     (* Zeilenende                    *)
 eof = fs;     (* Dateiende <ctrl-\>            *)
 csi = 233C;   (* command sequence introducer *)

END ASCII.
```

Anmerkung: Auf dem Amiga hat <line feed> (`lf`) die Funktion des Zeilenendes (`eol`).

Anmerkung: Das Zeichen `csi` (command sequence introducer) ist kein ASCII-Kontrollzeichen, sondern ein spezielles Bildschirmsteuerungs-Zeichen, das oft in Kontrollsequenzen wie für die Ausgabe auf das RAW:-Device → [DOS] benötigt wird.

Anmerkung: Es existiert *keine* Objekt-Datei zum Modul ASCII.

1 American Standard Code for Information Interchange

Assembler

Diese Modul hilft bei der Erstellung von INLINE-Code für MC68000 und MC68010. Es definiert für MC68000 Operationscodes und Adressierungsarten Konstanten. Diese können als Bausteine verwendet werden, um alle MC68000-Befehle zu erzeugen. Dafür ist die Kenntnis des Aufbau der Befehle nötig (→ [MC68000[).

Anmerkung: Einige Befehle sind nur auf MC68010 vorhanden! Für die genaue Handhabung sei auf [MC68000] verwiesen.

```
(*$M- *)
DEFINITION MODULE Assembler;

CONST
(* Register *)
 d0=0; d1=1; d2=2; d3=3; d4=4; d5=5; d6=6; d7=7;
 a0=0; a1=1; a2=2; a3=3; a4=4; a5=5; a6=6; a7=7;

(* Bits in Condition Code Register *)
 nbit=8; zbit=4; vbit=2; cbit=1;

(* Addressier-Modi *)
 ddir=00B; (* Dn *)
 adir=10B; (* An *)
 aidr=20B; (* (An) *)
 ainc=30B; (* (An)+ *)
 adec=40B; (* -(An) *)
 aoff=50B; (* d16(An) *)
 aidx=60B; (* d8(An,Rx) *)
 absW=70B; (* abs.W *)
 absL=71B; (* abs.L *)
 prel=72B; (* d16(PC) *)
 imm =74B; (* #n *)

(* Werte für Schiebeoperationen *)
 ls0=1; ls1=2; ls2=4; ls3=8; ls4=10H; ls5=20H; ls6=40H;
 ls7=80H; ls8=100H; ls9=200H; ls10=400H; ls11=800H;
 ls12=1000H; ls13=2000H; ls14=4000H; ls15=8000H;

(* MC68000 Instruktions-Mnemonics. *)
 abcdB=140400B; abcdmB=140410B;
 addB=150000B; addW=150100B; addL=150200B;
```

```
addmB=150400B; addmW=150500B; addmL=150600B;
addaW=150300B; addaL=150700B;
addiB=003000B; addiW=003100B; addiL=003200B;
addqB=050000B; addqW=050100B; addqL=050200B;
addxB=150400B; addxW=150500B; addxL=150600B;
addxmB=150410B; addxmW=150510B; addxmL=150610B;
andB=140000B; andW=140100B; andL=140200B;
andmB=140400B; andmW=140500B; andmL=140600B;
andiB=001000B; andiW=001100B; andiL=001200B;
andiCCR=001074B; andiSR=001174B;
aslB=160440B; aslW=160540B; aslL=160640B;
asliB=160400B; asliW=160500B; asliL=160600B;
aslmW=160700B;
asrB=160040B; asrW=160140B; asrL=160240B;
asriB=160000B; asriW=160100B; asriL=160200B;
asrmW=160300B;
bcc=062000B; bcs=062400B; beq=063400B;
bge=066000B; bgt=067000B; bhi=061000B;
ble=067400B; bls=061400B; blt=066400B; bmi=065400B;
bne=063000B; bpl=065000B; bvc=064000B; bvs=064400B;
bchg=000500B; bchgi=004100B;
bclr=000600B; bclri=004200B;
(* bkpt=044110B; *)
bra=060000B;
bset=000700B; bseti=004300B;
bsr=060400B;
btst=000400B; btsti=004000B;
chkW=040600B;
clrB=041000B; clrW=041100B; clrL=041200B;
cmpB=130000B; cmpW=130100B; cmpL=130200B;
cmpaW=130300B; cmpaL=130700B;
cmpiB=006000B; cmpiW=006100B; cmpiL=006200B;
cmpmB=130410B; cmpmW=130510B; cmpmL=130610B;
dbcc=052310B; dbcs=052710B; dbeq=053710B;
dbf=050710B; dbge=056310B;
dbgt=057310B; dbhi=051310B; dble=057710B;
dbls=051710B; dblt=056710B;
dbmi=055710B; dbne=053310B; dbpl=055310B;
dbt=050310B; dbvc=054310B;
dbvs=054710B;
dbra=050710B;
divsW=100700B;
divuW=100300B;
eorB=130400B; eorW=130500B; eorL=130600B;
eoriB=005000B; eoriW=005100B; eoriL=005200B;
eoriCCR=005074B; eoriSR=005174B;
exgAA=140510B; exgDA=140610B; exgDD=140500B;
extW=044200B; extL=044300B;
```

```
illegal=045374B;
jmp=047300B;
jsr=047200B;
lea=040700B;
linkW=047120B;
lslB=160450B; lslW=160550B; lslL=160650B;
lsliB=160410B; lsliW=160510B; lsliL=160610B;
lslmW=161700B;
lsrB=160050B; lsrW=160150B; lsrL=160250B;
lsriB=160010B; lsriW=160110B; lsriL=160210B;
lsrmW=161300B;
moveB=010000B; moveW=030000B; moveL=020000B;
moveCCRfr=041300B; moveCCRto=042300B;
moveSRfr=040300B; moveSRto=043300B;
moveUSPfr=047150B; moveUSPto=047140B;
moveaW=030100B; moveaL=020100B;
movecLfr=047172B; movecLto=047173B;
movemW=046200B; movemL=046300B;
movemmW=044200B; movemmL=044300B;
movepW=000410B; movepL=000510B;
movepmW=000610B; movepmL=000710B;
moveqL=070000B;
movesB=007000B; movesW=007100B; movesL=007200B;
mulsW=140700B;
muluW=140300B;
nbcdB=044000B;
negB=042000B; negW=042100B; negL=042200B;
negxB=040000B; negxW=040100B; negxL=040200B;
nop=047161B;
notB=043000B; notW=043100B; notL=043200B;
orB=100000B; orW=100100B; orL=100200B;
ormB=100400B; ormW=100500B; ormL=100600B;
oriB=000000B; oriW=000100B; oriL=000200B;
oriCCR=000074B;
oriSR=000174B;
pea=044100B;
reset=047160B;
rolB=160470B; rolW=160570B; rolL=160670B;
roliB=160430B; roliW=160530B; roliL=160630B;
rolmW=163700B;
rorB=160070B; rorW=160170B; rorL=160270B;
roriB=160030B; roriW=160130B; roriL=160230B;
rormW=163300B;
roxlB=160460B; roxlW=160560B; roxlL=160660B;
roxliB=160420B; roxliW=160520B; roxliL=160620B;
roxlmW=162700B;
roxrB=160060B; roxrW=160160B; roxrL=160260B;
roxriB=160020B; roxriW=160120B; roxriL=160220B;
```

```
 roxrmW=162300B;
 rtd=047164B; rte=047163B; rtr=047167B; rts=047165B;
 sbcdB=100400B;
 sbcdmB=100410B;
 scc=052300B; scs=052700B; seq=053700B; sf=050700B;
sge=056300B;
 sgt=057300B; shi=051300B; sle=057700B; sls=051700B;
 slt=056700B;
 smi=055700B; sne=053300B; spl=055300B; st=050300B;
svc=054300B;
 svs=054700B;
 stop=047162B;
 subB=110000B; subW=110100B; subL=110200B;
 submB=110400B; submW=110500B; submL=110600B;
 subaW=110300B; subaL=110700B;
 subiB=002000B; subiW=002100B; subiL=002200B;
 subqB=050400B; subqW=050500B; subqL=050600B;
 subxB=110400B; subxW=110500B; subxL=110600B;
 subxmB=110410B; subxmW=110510B; subxmL=110610B;
 swapW=044100B;
 tasB=045300B;
 trap=047100B; trapv=047166B;
 tstB=045000B; tstW=045100B; tstL=045200B;
 unlk=047130B;

END Assembler.
```

Um einen Befehl zu erstellen, müssen die benötigten Konstanten addiert werden. Da der Compiler Konstanten-Ausdrücke bereits zur Compilationszeit auswertet ("constant folding"), wird dadurch *kein* unnötiger Code erzeugt.

Untenstehendes Beispiel zeigt einige Anweisungen, die mit INLINE erzeugt wurden, gefolgt von der Ausgabe des Disassemblers.

```
INLINE(moveL+d0*ls9+prel,-58,
  addL+d0*ls9+imm,1234H,5678H,
  roliL+3*ls9+d0,
  moveL+aidr*ls3+a0+ls9+d0);
```

```
203A FFC6       MOVE.L   -58(PC)<FFFFFFF4>, D0
D0BC 12345678   ADD.L    #305419896, D0
E798            ROL.L    #3, D0
2280            MOVE.L   D0, (A1)
```

Anmerkung: Es existiert *keine* Objekt-Datei zum Modul Assembler.

Conversions

Das Modul `Conversions` stellt die beiden wichtigsten Prozeduren zur Umwandlung von ganzen Zahlen aus der lesbaren in die interne Darstellung und umgekehrt zur Verfügung. Dabei ist es möglich, sowohl vorzeichenbehaftete sowie vorzeichnenlose Grössen umzuwandeln.

```
DEFINITION MODULE Conversions;

PROCEDURE StrToVal(VAR str: ARRAY OF CHAR;
                   VAR l: LONGINT;
                   VAR signed: BOOLEAN; base: INTEGER;
                   VAR err: BOOLEAN);

PROCEDURE ValToStr(l: LONGINT; signed: BOOLEAN;
                   VAR str: ARRAY OF CHAR;
                   base, width: INTEGER;
                   fillChar: CHAR; VAR err: BOOLEAN);

END Conversions.
```

Die Prozedur `StrToVal` wandelt von der lesbaren zur internen Darstellung um. Die Zeichenkette `str` enthält eine Zahl zur Basis `base`. Diese Zahlenbasis kann jeden Wert zwischen 2 und 16 annehmen, die verwendeten Zeichen müssen allerdings innerhalb des entsprechend gültigen Bereiches [0..base-1] sein, also beispielsweise zwischen 0 und 7 für Zahlen in Oktaldarstellung. Die Zeichenkette darf insbesondere auch keine Leerzeichen enthalten, das Nullzeichen (0C) als Abschlusszeichen ist selbstverständlich erlaubt. Das Resultat wird im LONGINT-Parameter `l` abgelegt. Ein Fehler - angezeigt durch den Parameter `err` - kann durch folgende Ereignisse verursacht werden:

- Die Zahlenbasis ist ausserhalb des erlaubten Bereichs.
- Die Eingabe-Zeichenkette enthält Zeichen, die nicht zur angegebenen Basis passen.
- Der resultierende Wert kann nicht in einem LONGINT gespeichert werden.

Ist der Resultatswert `l` negativ und hat `signed` gleichzeitig den Wert `FALSE`, so wurde eine Zahl übergeben, die kein Minuszeichen enthält, deren Wert aber nicht in einer LONGINT-Zahl abgespeichert werden kann. In diesem Fall ist die Zahl `l` in einem LONGCARD abzuspeichern. Entweder wird ein Varianten-Verbund übergeben, oder man muss sich mit einer CAST-Anweisung (→ SYSTEM) behelfen.

Die umgekehrte Umwandlung wird durch die Prozedur `ValToStr` durchgeführt. Die im LONGINT-Parameter `l` abgelegte Zahl wird in die Zeichenkette `str` umgewandelt und formatiert. Die Zahl wird in einem Feld der Grösse `width` (Absolutbetrag) positioniert. Das Vorzeichen von `width` bestimmt die Anordnung der Zahl im Feld. Bei einem positiven Wert von `width` wird die Zahl im Feld rechtsbündig, mit führenden Füllzeichen `fillChar`, sonst linksbündig mit nachfolgenden Füllzeichen, angeordnet.

Für das Vorzeichen von `l` und dem Wert von `signed` gilt die gleiche Konvention wie bei `StrToVal`. Zahlen und Zeichenketten, die nur mit `Conversions` bearbeitet werden, benötigen daher keine weitere Bearbeitung.

Beispiele:

```
str:="03F9";
large:="3000000000";                    (* >2^31 *)
StrToVal(str,number,signed,16,err);
  (* number:      1017
     signed:      FALSE
     err: FALSE *)
StrToVal(str,number,signed,10,err);
  (* number:      undefiniert
     signed:      undefiniert
     err: TRUE (Basis) *)
StrToVal(large,number,signed,10,err);
  (* number:      -1294967296
     signed:      FALSE
     err: FALSE *)

ValToStr(27,FALSE,str,8,5,'*',err);
  (* str: "***33"
     err: FALSE *)
ValToStr(45054,FALSE,str,16,-10,' ',err);
  (* str: "AFFE      "
     err: FALSE *)
```

Anmerkung: Die interne Darstellung (Zweierkomplement) von 3'000'000'000 (LONGCARD) ist gleich der internen Darstellung von -1'294'967'296 (LONGINT).

Coroutines

Das Modul erlaubt die Handhabung von Coroutinen. Es bietet dieselben Prozeduren an, die das Modul SYSTEM in früheren Modula-2 Versionen zur Verfügung stellte und folgt der Definition aus [PIM3].

```
DEFINITION MODULE Coroutines;

FROM SYSTEM IMPORT
 ADDRESS;

TYPE
 PROCESS=ADDRESS;

PROCEDURE TRANSFER(VAR source, destination: PROCESS);
PROCEDURE NEWPROCESS(p: PROC; wsp: ADDRESS; size:
LONGINT;
                     VAR new: PROCESS);

END Coroutines.
```

Prozedur NEWPROCESS ruft eine neue Coroutine ins Leben, wobei die Prozedur `proc` diese neue Coroutine bildet. Die Variablen `wsp` und `size` bezeichnen den zur Verfügung gestellten Arbeitsspeicher, erstere gibt die Adresse, letztere die Grösse des Arbeitsspeichers. Die Variable `new` charakterisiert die neu kreierte Coroutine. Die Coroutine wird nicht gestartet, sondern nur initialisiert.

Der Arbeitsspeicher muss alle lokalen Variablen der Coroutine und der von ihr aufgerufenen Prozeduren beinhalten. Es können daher keine allgemeingültigen Werte für die optimale Grösse des Arbeitsspeichers angegeben werden.

TRANSFER suspendiert die durch `source` charakterisierte Coroutine, während mit `dest` an der Stelle der Suspension fortgefahren wird. Wird `source` wieder fortgesetzt, wird die unmittelbar auf TRANSFER folgende Anweisung ausgeführt.

Anmerkung: Der Typ PROCESS existiert nur aus Gründen der Kompatibilität und der Klarheit. Falls bevorzugt, kann er überall durch ADDRESS ersetzt werden.

FFPConversions

Das Modul FFPConversions stellt die Prozeduren zur Umwandlung von Gleitpunkt-Zahlen (Dezimalbrüchen) zwischen interner und lesbarer Darstellung und umgekehrt zur Verfügung. Dabei wird das spezielle FFP-Format verwendet.

```
DEFINITION MODULE FFPConversions;

FROM SYSTEM IMPORT
 FFP;

PROCEDURE StrToReal(VAR s: ARRAY OF CHAR; VAR r: FFP;
                    VAR err: BOOLEAN);

PROCEDURE RealToStr(r: FFP; VAR s: ARRAY OF CHAR;
                    m, n: INTEGER; expo: BOOLEAN;
                    VAR err: BOOLEAN);

END FFPConversions.
```

Die Prozedur StrToReal wandelt die in str gespeicherte Gleitpunktzahl in die interne FFP-Darstellung um. Enthält die Zeichenkette nicht erlaubte Zeichen, oder kann die beschriebene Zahl nicht in einer FFP-Variablen dargestellt werden, erhält err den Wert TRUE.

Die Prozedur RealToStr erlaubt vielseitige Umwandlungen von r in lesbare Darstellungsformen. Die resultierende Zeichenkette enthält genau m Zeichen, oder soviele wie s Platz bietet, falls m zu gross ist. Ist m positiv, werden notwendige Leerzeichen vor der Zahl angefügt, sonst werden sie angehängt. Die Anzahl Zeichen hinter dem Dezimalpunkt wird durch n gegeben. Falls in Berücksichtigung zur Feldgrösse die Anzahl der Dezimalstellen zu gross gewählt wurde, wird diese auf das entsprechende Maximum gekürzt. Ist n=0 wird kein Dezimalpunkt geschrieben. Der Parameter expo gibt an, ob die wissenschaftliche Schreibweise (Exponentialdarstellung) gewünscht ist.

Beispiele:
```
r:=12.976
RealToStr(r,str,5,2,FALSE,err);
  (* str: "12.98"
     err: FALSE *)
RealToStr(r,str,4,2,FALSE,err);
  (* str: "13.0"
     err: FALSE *)
RealToStr(r,str,3,2,FALSE,err);
  (* str: " 13"
     err: FALSE *)
RealToStr(r,str,2,2,FALSE,err);
  (* str: "13"
     err: FALSE *)
RealToStr(r,str,1,2,FALSE,err);
  (* str: undefiniert
     err: TRUE *)
RealToStr(r,str,-8,2,FALSE,err);
  (* str: "12.98   "
     err: FALSE *)
RealToStr(r,str,10,2,TRUE,err);
  (* str: "  1.30E+01"
     err: FALSE *)
```

Mögliche Fehlerquellen (`err=TRUE`) sind ein negativer Wert von `n`, der Wert Null für `m` oder `s` beinhaltet zu wenig Zeichen, um die resultierende Zeichenkette abzuspeichern.

FFPInOut

Dem Standardmodul `RealInOut` entsprechendes Modul für die Ein- und Ausgabe von Zahlen des Typs `FFP`. Der Ein- und Ausgabestrom wird vom Modul `InOut` kontrolliert.

```
DEFINITION MODULE FFPInOut;

FROM SYSTEM IMPORT
 FFP;

VAR
 done:BOOLEAN;

PROCEDURE WriteReal(r: FFP; m,n: INTEGER);

PROCEDURE ReadReal(VAR r: FFP);

END FFPInOut;
```

Die Prozedur `WriteReal` schreibt die übergebene Zahl `r` in ein Feld der Grösse `m`, wobei im Maximum `n` Dezimalstellen geschrieben werden. Füllt die Darstellung das Feld nicht aus, wird das Feld mit Leerzeichen ausgefüllt. Ist die Feldgrösse positiv, werden die Leerzeichen vor die Zahl gesetzt, sonst werden sie angehängt.

`ReadReal` liest eine Zeichenkette von der Eingabe (InOut) und wandelt diese in die FFP-Zahl `r` um.

FileMessage

FileMessage liefert lesbare Fehlermeldungen aus den Statuswerten des Moduls `FileSystem`.

```
DEFINITION MODULE FileMessage;

FROM FileSystem IMPORT
 Response;

TYPE
 StrPtr = POINTER TO ARRAY [0..255] OF CHAR;

PROCEDURE ResponseText(res: Response; VAR textPtr:
StrPtr);

END FileMessage.
```

Die Prozedur `ResponseText` liefert aus dem Dateistatus `res` in `textPtr` einen Zeiger auf einen Text, der unmittelbar ausgegeben werden kann.

Aufruf:

```
VAR
  f: File;
  p: StrPtr;
  ...
  ResponseText(f.res, p);
  WriteString(p^); WriteLn;
  ...
```

Anmerkung: `ResponseText` gibt die Adresse der Zeichenkette im Konstantenbereich zurück, deshalb darf diese Zeichenkette nicht verändert werden! Die Zeichenkette darf auch nicht auf 256 Zeichen aufgefüllt werden.

FileNames

Modul für die Behandlung von Dateinamen-Eingaben.

```
DEFINITION MODULE FileNames;

TYPE
 ReadProc=PROCEDURE(VAR CHAR);
 WriteProc=PROCEDURE(CHAR);

CONST
 noEcho=WriteProc(NIL);

PROCEDURE ReadFileName(VAR fname: ARRAY OF CHAR;
                       VAR len: INTEGER;
                       readProc: ReadProc;
                       writeProc: WriteProc);

PROCEDURE GetPath(VAR fname, path: ARRAY OF CHAR;
                  VAR len: INTEGER);

PROCEDURE GetExtension(
           VAR fname, extension: ARRAY OF CHAR;
           VAR len: INTEGER);

PROCEDURE MakeFileName(
           VAR fname: ARRAY OF CHAR;
           VAR len: INTEGER;
           dev, dir, name, ext: ARRAY OF CHAR);

END FileNames.
```

`ReadFileName` liest einen Dateinamen mit den übergebenen Ein- und Ausgabe-Prozeduren. `len` gibt die Anzahl gelesenen Zeichen. Alle Zeichen bis 176C werden akzeptiert.

```
ReadFileName(fname, len, Terminal.Read, noEcho);
```
Benutzereingabe: `df1:exec/sym/Alerts.sym` <RETURN>
```
fname="df1:exec/sym/Alerts.sym"; len=23;
```

`GetPath` löscht die Pfadinformation aus `fname` und speichert diese in `path`. `len` wird entsprechend dem neuen Dateinamen gesetzt.

`GetExtension` arbeitet ähnlich wie `GetPath`, ausser dass die Erweiterung gelöscht wird. Als Erweiterung wird angenommen, was hinter dem letzten Punkt steht.

`MakeFileName` verkettet die 4 übergebenen Zeichenketten aneinander und speichert den gesamten Dateinamen in `fname`.

FileSystem

Das Standardmodul FileSystem bietet Prozeduren zur Dateibehandlung an. Es ermöglicht den sequentiellen, sowie den wahlweisen Zugriff auf Dateien. Lese- und Schreiboperationen können beliebig gemischt werden. Dateien, die mit dem Modul FileSystem bearbeitet werden, werden beim Programmabsturz oder -ende noch auf der Diskette vervollständigt und geschlossen.

Die Anzahl der gleichzeitig verwendeten Dateien ist nur durch das Betriebssystem beschränkt. Im Gegensatz zu AmigaDOS (→ [DOS] ist der Dateizugriff durch `FileSystem` gepuffert. Dies kann den Zugriff bis zu Faktor 10 verschnellern. Die Puffergrösse muss beim Eröffnen der Datei festgelegt werden.

```
DEFINITION MODULE FileSystem;

FROM SYSTEM IMPORT
 ADDRESS,BYTE;
FROM Dos IMPORT
 FileHandlePtr;

TYPE
 CHARPtr;
 Response=(
  done,notdone,lockErr,openErr,readErr,writeErr,
  seekErr,memErr,inUse,notFound,diskWriteProtected,
  deviceNotMounted,diskFull,deleteProtected,
  writeProtected,notDosDisk,noDisk
 );
 FileMode=(read,write,noBuffer);
 FileModeSet=SET OF FileMode;
 File=RECORD
  bufa,ela,ina,topa: CHARPtr;
  bufPos,filePos: LONGINT;
  file: FileHandlePtr;
  mode: FileModeSet;
  eof: BOOLEAN;
  res: Response;
 END;

PROCEDURE Lookup(VAR f: File; name: ARRAY OF CHAR;
                 bufferSize: CARDINAL; new: BOOLEAN);
PROCEDURE Close(VAR f: File);
```

```
PROCEDURE Delete(VAR f: File);
PROCEDURE SetPos(VAR f: File; pos: LONGINT);
PROCEDURE GetPos(VAR f: File; VAR pos: LONGINT);
PROCEDURE Length(VAR f: File; VAR pos: LONGINT);
PROCEDURE ReadChar(VAR f: File; VAR ch: CHAR);
PROCEDURE WriteChar(VAR f: File; ch: CHAR);
PROCEDURE ReadByteBlock(VAR f: File;
                        VAR block: ARRAY OF BYTE);
PROCEDURE WriteByteBlock(VAR f: File;
                         VAR block: ARRAY OF BYTE);
PROCEDURE ReadBytes(VAR f: File; adr: ADDRESS;
                    len: LONGINT; VAR actual: LONGINT);
PROCEDURE WriteBytes(VAR f: File; adr: ADDRESS;
                     len: LONGINT; VAR actual: LONGINT);

END FileSystem.
```

Eine Datei wird mit dem strukturierten Typ `File` idenifiziert. Für den Anwender sind davon jedoch nur die Felder `res` und `eof` von Bedeutung. Das Feld `res` enthält das Resultat der zuletzt ausgeführten Dateimanipulation, während `eof` das Dateiende anzeigt. Das Resultatsfeld sollte nach jedem Aufruf einer Prozedur aus `FileSystem` kontrolliert werden, die Prüfung auf das Dateiende muss.nur nach Lesezugriffen erfolgen. Die verbleibenden Felder verwalten den Puffer und stellen die Verbindung zu AmigaDOS her.

Der Aufzählungstyp `Response` enthält die möglichen Zustände, die nach einem Dateizugriff auftreten können. Im Normalfall enthält das Feld `res` den Wert `done`, was bedeutet, dass die letzte Operation erfolgreich ausgeführt wurde. Nebst der allgemeinen Fehlermeldung `notDone` enthält `Response` einige AmigaDOS-spezifische Meldungen.

Um die Behandlung von Fehlersituationen zu vereinfachen, gilt folgende Regel: Alle Prozeduren ausser `Lookup` und `Close` versuchen die Operation nur auszuführen, falls das Feld `res` gleich dem Wert `done` ist. So ist es möglich, eine Datei vollständig zu lesen oder zu schreiben, ohne jedesmal auf einen Fehler testen zu müssen. `Lookup` und `Close` sind aus folgenden Gründen ausgenommen:

- `Lookup` kann keine definierten Werte in der `File`-Variablen erwarten, da erst der Aufruf dieser Prozedur die Variable mit gültigen Werten belegt.

- `Close` muss versuchen, eine Datei auch dann zu schliessen, wenn zuvor ein Fehler aufgetreten ist.

Es dürfen keine Operationen ausser `Lookup` auf nichtinitialisierte Filevariablen ausgeführt werden.

Beispiel:

```
VAR
 f: FILE; ch: CHAR;
BEGIN
 Lookup(f,"Datei",1024,FALSE);
 Assert(f.res=done,"Lookup misslungen!");
 LOOP
  ReadChar(f,ch);
  IF f.eof OR (f.res#done) THEN
   EXIT
  END
  Bearbeite(ch)
 END;
 (* Close auf jeden Fall aufrufen *)
 Close(f)
END
```

Mit der Prozedur `Lookup` wird eine Datei zum Lesen und/oder zum Schreiben eröffnet. Der Parameter `name` enthält den Namen der zu eröffnenden Datei. Mit `new` kann angegeben werden, ob eine neue oder eine bereits bestehende Datei eröffnet werden soll. Hat `new` den Wert `TRUE` und es existiert eine Datei mit dem angegebenen Namen, so wird diese beim Eröffnen gelöscht. Die Filevariable wird dem Ausgang der Operation entsprechend initialisiert.

Die Daten einer Datei werden zwischengespeichert. So resultieren mehrere Lesebefehle mit nur einem Lesebefehl auf der Datei, und Schreibbefehle werden erst auf der Diskette ausgeführt, wenn der zur Pufferung verwendete Puffer überläuft. Die Grösse des verwendeten Puffers wird beim `Lookup`-Befehl mit dem Parameter `buffer` gesteuert. Wird keine Pufferung gewünscht, kann auch 0 übergeben werden.

`Close` schliesst eine zuvor mit `Lookup` eröffnete Datei. Noch im Puffer verbliebene Daten werden vorher noch auf die Datei geschrieben.

`Delete` löscht die angebene Datei. Damit der Parameter initialisiert ist, muss die Datei zuerst geöffnet werden.

Mit `SetPos` kann eine beliebige Position innerhalb der Datei gesprungen werden. Es kann nicht über das (aktuelle) Dateiende positioniert werden; ein solcher Aufruf führt zu einem Fehler. Nach einem Fehler ist der innere Zustand einer Filevariablen undefiniert. Es ist im allgemeinen nicht möglich, mit dieser Datei weiter zu arbeiten. Das Feld `res` darf nicht auf `done` zurückgesetzt werden.

`GetPos` gibt die aktuelle Position innerhalb der Datei zurück.

`Length` gibt die aktuelle Dateilänge zurück. Dies ist zugleich der höchste zulässige Wert, der bei `SetPos` verwendet werden darf.

`ReadChar` und `WriteChar` sind die Prozeduren zum Lesen und Schreiben von einzelnen Zeichen.

`ReadByteBlock` und `WriteByteBlock` sind Prozeduren, die vor allem eingesetzt werden sollen, wenn beliebige Variablen eines strukturierten Typs gelesen oder geschrieben werden sollen.

`ReadBytes` und `WriteBytes` arbeiten ähnlich wie `ReadByteBlock` und `WriteByteBlock`. Der Datenbereich wird hier über die Adresse `adr` und die Längenangabe `length` identifiziert. Der Parameter `actual` gibt an, wieviele Byte gelesen beziehungsweise geschrieben werden konnten.

Erkennen des Dateiendes bei Leseoperationen

Das Feld `eof` erhält den Wert `TRUE`, wenn eine Leseoperation genau hinter dem letzten gültigen Byte der Datei gestartet wird. Eine Leseoperation, die `eof` setzt, kann deshalb nie gültige Daten von der Datei lesen, obwohl `res` auf dem Wert `done` verbleibt. Wenn das Dateiende während einer Leseoperation (`ReadBytes` oder `ReadByteBlock`) erreicht wird, bleibt `eof` auf `FALSE`. Der Resultatswert `res` einer solchen Leseoperation ist abhängig von der verwendeten Methode.

`ReadBytes:` Da ReadBytes die Anzahl der gelesen Bytes immer angibt, können beim Dateiende zwei Fälle auftreten:

1. In der Datei sind zwischen 1 und Blockgrösse Bytes bis zum Dateiende vorhanden. Diese werden in den Block eingelesen und die aktuelle Länge wird entsprechend gesetzt.

```
eof = FALSE
res = done
```

2. Die Datei wurde bereits vollständig gelesen. Das Dateiende ist erreicht, `eof` ist wahr und der Lesezugriff ist erfolgreich.

```
eof = TRUE
res = done
```

`ReadByteBlock:` Diese Prozedur kennt keine Möglichkeit, unvollständige Blöcke anzuzeigen. Dadurch ergibt sich ein von `ReadBytes` unterschiedliches Verhalten:

1. In der Datei sind zwischen 1 und Blockgrösse-1 Byte bis zum Dateiende vorhanden. Ein unvollständiger Block ist jedoch kein gültiges Resultat dieser Operation, der Inhalt von Block ist undefiniert.

```
eof = FALSE
res = notdone
```

2. Die Datei wurde bereits vollständig gelesen. Das Dateiende ist erreicht, `eof` ist wahr und der Lesezugriff ist erfolgreich.

```
eof = TRUE
res = done
```

Heap

Das Modul Heap stellt die für dynamische Speicher-Allozierung und Deallozierung benötigten Prozeduren zur Verfügung. Es behält die Kontrolle über alle allozierten Speicherbereiche, und gibt sie im Falle eines Programmabsturzes oder am Programmende wieder frei.

```
DEFINITION MODULE Heap;

FROM SYSTEM IMPORT
 ADDRESS;

PROCEDURE Allocate(VAR adr: ADDRESS; size: LONGINT);
PROCEDURE Deallocate(VAR adr: ADDRESS);

PROCEDURE Available(chipMem: BOOLEAN): LONGINT;
PROCEDURE Largest(chipMem: BOOLEAN): LONGINT;

(* spezielle Funktionen *)
PROCEDURE AllocMem(VAR adr: ADDRESS; size: LONGINT;
                   chipMem: BOOLEAN);

PROCEDURE Alloc(VAR adr: ADDRESS; size: LONGINT;
                chipMem: BOOLEAN; level: INTEGER);

PROCEDURE BlockSize(adr: ADDRESS): LONGINT;

END Heap.
```

Das Modul Heap ist eine einfache Schnittstelle zu den Funktionen der Speicherverwaltung des Betriebsystem. Es werden die Exec-Funktionen (→ Abschnitt Exec) `AllocMem`, `FreeMem` und `AvailMem` verwendet. Die durch Prozeduren aus Heap angeforderten Speicherblöcke werden verkettet. Dies ermöglicht später die Freigabe der allozierten Speicherblöcke. Diese Verkettung benötigt etwas zusätzlichen Speicher und Zeit. Die Prozeduren `Alloc` und `Blocksize` sind auf diese Zusatzinformation angewiesen.

Folgende Informationen werden unmittelbar vor dem allozierten Block abgelegt:

```
next: ADDRESS;    (* Zeiger auf nächsten Block oder NIL *)
level: Byte;      (* Level, auf dem der Block alloziert
                     wurde *)
size: [1..2^24];  (* Grösse des Blocks *)
```

Diese Zusatzinformationen benötigen ingesamt 8 Byte. Die Funktion des Level-Elements ist im Kapitel des Laufzeitsystems beschrieben.

Die Prozedur `Allocate` alloziert einen Bereich der in Byte gegebenen Grösse `size` und weist dessen Adresse dem Parameter `adr` zu. Wird eine Grösse von 2^{24} oder mehr Byte (>=16 MByte) angefordert, bricht das aufrufende Programm mit einem Laufzeitfehler ab. Ist nicht mehr genügend Speicher vorhanden, enthält `adr` den Wert `NIL`. Ein erfolgreich angeforderter Speicherblock wird mit Nullen initialisiert.

`Deallocate` gibt einen vorher mit `Allocate`, `AllocMem` oder `Alloc` reservierten Speicherbereich wieder frei und weist dem gegebenen Zeiger den Wert `NIL` zu. Weist der Zeiger auf keinen vorher durch `Allocate`, `AllocMem` oder `Allocate` reservierten Speicher, wird der Aufruf ignoriert.

`Available` gibt die Grösse des gesamten freien Speicherplatzes zurück. Ist der Parameter `chipMem` auf `FALSE` gesetzt, wird die Speichergrösse in Chip- und Fast-Memory zusammen, sonst nur diejenige in Chip-Memory zurückgegeben. Dieser Speicherplatz ist allerdings nicht als zusammenhängender Bereich erhältlich, sondern setzt sich aus beliebigen Blöcken zusammen.

`Largest` gibt die Grösse des grössten, aneinanderhängenden Speicherbereiches zurück. Für den Parameter `chipMem` gilt das gleiche wie bei `Available`.

Die Prozeduren `Alloc` und `AllocMem` sind Varianten der `Allocate`-Prozedur. Der Parameter `chipMem` gibt an, ob der Speicher im Chip- oder im Fast-Memory (→ [RKM1]) alloziert werden soll. Ist der Wert `TRUE`, wird der Speicher im Chip-Memory alloziert,

ist der Wert `FALSE`, wird der Speicher nur dann im Chip-Memory alloziert, falls im Fast-Memory zu wenig oder gar kein Platz ist. In jedem Fall wird der Speicherbereich mit Nullen initialisiert.

Die Prozedur `Alloc` erlaubt zusätzlich die Angabe der Level-Information, während ansonsten das Resultat der Funktion `Arts.CurrentLevel()` eingesetzt wird.

BlockSize ist eine Hilfsfunktion, die es erlaubt, nachträglich die Grösse eines Speicherbereichs abzufragen. Wurde der Bereich durch keine der Funktionen `Allocate`, `AllocMem` und `Alloc` angefordert, wird der Wert 0 zurückgegeben.

Warnung:	Greifen Sie nie auf einen Zeiger zu, dessen Speicherbereich nicht alloziert oder bereits dealloziert wurde! Ein Absturz ist damit garantiert! Verwenden Sie `Storage` an Stelle von `Heap`, wenn Sie keine Ausnahmebehandlungen vorsehen wollen.

Anmerkung: Aus Gründen der Portabilität sollte wenn möglich `Allocate` den Prozeduren `AllocMem` und `Alloc` vorgezogen werden.

InOut

Standard-Modul für formatierte Ein- und Ausgabe auf Bildschirm oder auf Dateien. Die primäre Ein- und Ausgabe bezieht sich auf die Ein- und Ausgabeströme wie im Kapitel Terminal beschrieben, kann jedoch auf Dateien umgelenkt werden. Dieses Umlenken ist unabhängig von der Umlenkmöglichkeit von AmigaDOS (→ [DOS])

```
DEFINITION MODULE InOut;
FROM SYSTEM IMPORT
 BYTE;
IMPORT ASCII;

CONST
 eol = ASCII.eol;
VAR
 done:BOOLEAN;
 termCh:CHAR;

PROCEDURE OpenInput(defExt: ARRAY OF CHAR);
PROCEDURE OpenOutput(defExt: ARRAY OF CHAR);

PROCEDURE CloseInput;
PROCEDURE CloseOutput;

PROCEDURE Read(VAR ch: CHAR);
PROCEDURE ReadString(VAR str:  ARRAY OF CHAR);
PROCEDURE ReadInt(VAR x:INTEGER);
PROCEDURE ReadLongInt(VAR x: LONGINT);
PROCEDURE ReadCard(VAR x: CARDINAL);
PROCEDURE ReadLongCard(VAR x: LONGCARD);
PROCEDURE ReadBytes(VAR blk: ARRAY OF BYTE);

PROCEDURE Write(ch: CHAR);
PROCEDURE WriteLn;
PROCEDURE WriteString(str: ARRAY OF CHAR);
PROCEDURE WriteInt(x: LONGINT; n: CARDINAL);
PROCEDURE WriteCard(x: LONGCARD; n: CARDINAL);
PROCEDURE WriteOct(x: LONGINT; n: CARDINAL);
PROCEDURE WriteHex(x: LONGINT; n: CARDINAL);
PROCEDURE WriteBytes(VAR blk: ARRAY OF BYTE);

END InOut.
```

`OpenInput` und `OpenOutput` lesen beide einen Dateinamen von der Tastatur (Aufruf von `Terminal.Read`, → Abschnitt Terminal) und öffnen eine Ein-, beziehungsweise eine Ausgabedatei. Die Variable `done` enthält nach erfolgreicher Eröffnung den Wert `TRUE`. Endet der eingegebene Name mit einem Punkt ("."), wird die Zeichenkette `defExt` an den Dateinamen angehängt.

`CloseInput` und `CloseOutput` schliessen eine zuvor eröffnete Ein- oder Ausgabedatei. Der Eingabestrom kommt von der Tastatur und die Ausgabe geschieht wieder über den Bildschirm (`Terminal.Read`, `Terminal.Write`).

`ReadString` liest eine Zeichenkette ein, d.h eine Folge von Zeichen, die weder Leerzeichen noch Kontrollzeichen enthält. Führende Leerzeichen werden ignoriert. Die Eingabe wird durch jedes Kontrollzeichen oder ein Leerzeichen (≤ " ") beendet. Soll ein Leerzeichen in die Zeichenkette eingefügt werden, muss dieses in Anführungszeichen (") gesetzt werden (vgl. Arguments).

`ReadInt`, `ReadCard`, `ReadLongInt` und `ReadLongCard` lesen eine Zeichenkette ein und wandeln diese entsprechend um. Führende Leerzeichen werden ignoriert.

`ReadBytes` kann nur angewendet werden, wenn eine Eingabedatei eröffnet wurde. Es werden soviele Byte gelesen, wie `blk` Platz bietet.

`WriteInt`, `WriteCard`, `WriteOct` und `WriteHex` schreiben die übergebene Zahl `x` mit mindestens `n` Zeichen. Werden nicht alle `n` Zeichen benötigt, werden der Zahl Leerzeichen (`WriteInt`, `WriteCard`) beziehungsweise Nullen (`WriteOct`, `WriteHex`) vorangestellt.

`WriteBytes` schreibt den gegebenen Block auf eine zuvor eröffnete Ausgabedatei.

Die Variable `termCh` enthält das Abschlusszeichen nach einem Aufruf der Prozeduren `ReadString`, `ReadInt`, `ReadLongInt`, `ReadCard` und `ReadLongCard`.

Anmerkung: Solange Ein- und Ausgabe nicht explizit umgelenkt wurde (Aufruf von `OpenInput` oder `OpenOutput`) wird diese über das Modul Terminal abgewickelt. Für Informationen über die in Terminal verwendeten Ein- und Ausgabekanäle muss im Abschnitt Terminal nachgeschlagen werden.

Anmerkung: Während `INTEGER`-Werte (`CARDINAL`) durch die gleiche Prozedur geschrieben werden können wie `LONGINT`-Werte (`LONGCARD`), wird für deren Eingabe eine eigene Prozedur benötigt, da bei Variablen-Parameter die formalen und die aktuellen Parameter typenkompatibel sein müssen. `INTEGER` (`CARDINAL`) und `LONGINT` (`LONGCARD`) sind jedoch nur zuweisungskompatibel.

Anmerkung: Um die Wahl zwischen Bildschirmausgabe und Dateiausgabe wählbar zu halten, sollte nach dem Aufruf von `OpenOutput` unbedingt die Variable `termCh` abgefragt werden. Enthält die Variable `termCh` das Nullzeichen (0C), wurde ein leerer Name eingegeben. Das Programm sollte dann normal fortgesetzt werden.

LongRealConversions

Das Modul `LongRealConversions` stellt die Prozeduren zur Umwandlung von Gleitpunkt-Zahlen grosser Genauigkeit (Dezimalbrüche) zwischen interner und lesbarer Darstellung und umgekehrt zur Verfügung.

```
DEFINITION MODULE LongRealConversions;

PROCEDURE StrToReal(VAR s: ARRAY OF CHAR;
                    VAR r: LONGREAL;
                    VAR err: BOOLEAN);

PROCEDURE RealToStr(r: LONGREAL; VAR s: ARRAY OF CHAR;
                    m, n: INTEGER; expo: BOOLEAN;
                    VAR err: BOOLEAN);

END LongRealConversions.
```

Die Prozedur `StrToReal` wandelt die in `str` gespeicherte Gleitpunktzahl in die interne LONGREAL-Darstellung um. Enthält die Zeichenkette nicht erlaubte Zeichen, oder kann die beschriebene Zahl nicht in einer REAL-Variablen dargestellt werden, erhält `err` den Wert TRUE.

Die Prozedur `RealToStr` erlaubt vielseitige Umwandlungen von `r` in lesbare Darstellungsformen. Die resultierende Zeichenkette enthält genau `m` Zeichen, oder soviele wie `s` Platz bietet, falls `m` zu gross ist. Ist `m` positiv, werden notwendige Leerzeichen vor der Zahl angefügt, sonst werden sie angehängt. Die Anzahl Zeichen hinter dem Dezimalpunkt wird durch `n` gegeben. Falls in Berücksichtigung zur Feldgrösse die Anzahl der Dezimalstellen zu gross gewählt wurde, wird diese auf das entsprechende Maximum gekürzt. Ist `n=0` wird kein Dezimalpunkt geschrieben. Der Parameter `expo` gibt an, ob die wissenschaftliche Schreibweise (Exponentialdarstellung) gewünscht ist.

Beispiele:

```
r:=12.976
RealToStr(r,str,5,2,FALSE,err);
  (* str:  "12.98"
     err:  FALSE *)
RealToStr(r,str,4,2,FALSE,err);
  (* str:  "13.0"
     err:  FALSE *)
RealToStr(r,str,3,2,FALSE,err);
  (* str:  " 13"
     err:  FALSE *)
RealToStr(r,str,2,2,FALSE,err);
  (* str:  "13"
     err:  FALSE *)
RealToStr(r,str,1,2,FALSE,err);
  (* str:  undefiniert
     err:  TRUE *)
RealToStr(r,str,-8,2,FALSE,err);
  (* str:  "12.98   "
     err:  FALSE *)
RealToStr(r,str,10,2,TRUE,err);
  (* str:  "  1.30E+01"
     err:  FALSE *)
```

Mögliche Fehlerquellen (`err=TRUE`) sind ein negativer Wert von `n`, der Wert Null für `m` oder `s` beinhaltet zu wenig Zeichen, um die resultierende Zeichenkette abzuspeichern.

LongRealInOut

Dem Standardmodul `RealInOut` entsprechendes Modul für die Ein- und Ausgabe von Zahlen des Typs LONGREAL. Der Ein- und Ausgabestrom wird vom Modul `InOut` kontrolliert.

```
DEFINITION MODULE LongRealInOut;

VAR
 done:BOOLEAN;

PROCEDURE WriteReal(r:LONGREAL; m,n:INTEGER);

PROCEDURE ReadReal(VAR r:LONGREAL);

END LongRealInOut;
```

Die Prozedur `WriteReal` schreibt die übergebene Zahl `r` in ein Feld der Grösse `m`, wobei im Maximum `n` Dezimalstellen geschrieben werden. Füllt die Darstellung das Feld nicht aus, wird das Feld mit Leerzeichen ausgefüllt. Ist die Feldgrösse positiv, werden die Leerzeichen vor die Zahl gesetzt, sonst angehängt.

`ReadReal` liest eine Zeichenkette von der Eingabe (InOut) und wandelt sie in die LONGREAL-Zahl `r` um.

MathLib0

Standard Modul für einige transzedente Funktionen. Weitere Funktionen sind direkt auf diesen aufbaubar.

```
DEFINITION MODULE MathLib0;

CONST
 e=2.7182818284;
 pi=3.1415926536;

PROCEDURE sqrt(x:REAL):REAL;
PROCEDURE exp(x: REAL):REAL;
PROCEDURE ln(x:REAL):REAL;
PROCEDURE sin(x:REAL):REAL;
PROCEDURE cos(x:REAL):REAL;
PROCEDURE arctan(x:REAL):REAL;

END MathLib0.
```

Die Einheit der Argumente für `sin` und `cos` sowie des Resultats von `arctan` ist Bogenmass (Radiant Abk. rad). Dabei gilt folgende Formel:

$$1 \text{ rad} = \frac{180°}{\pi}$$

`ln` gibt den natürlichen Logarithmus (Basis `e`) von `x` zurück.

Warnung:	Funktionsrufe mit Argumenten, deren Resultatwert nicht definiert ist (beispielsweise Quadratwurzel oder Logarithmus einer negativen Zahl), führen zum Abbruch des Programms.

Beispiele für aufbauende Funktionen:

```
PROCEDURE tan(x: REAL): REAL;
BEGIN
  RETURN sin(x)/cos(x)
END tan;
```

```
PROCEDURE log(x: REAL): REAL;
BEGIN
  RETURN ln(x) / ln(10)
END log;

PROCEDURE RadToAngle(rad: REAL); REAL;
BEGIN
  RETURN rad * 180 / pi
END RadToAngle;

PROCEDURE AngleToRad(angle: REAL); REAL;
BEGIN
  RETURN angle * pi / 180
END RadToAngle;
```

Die analogen Beispiele gelten auch für die Moduln MathLibFFP und MathLibLong. Da die Liste der bekanntesten transzedenten Funktionen bereits sehr gross ist, selten jedoch wirklich viele gebraucht werden, wurde auf die Definition weiterer MathLib...-Moduln verzichtet. Obige Beispiele zeigen, dass die Definition zusätzlicher Moduln keine Schwierigkeiten bereiten sollte.

MathLibFFP

Modul, das zur Anwendung kommt, wenn aus Gründen der Portabilität das Modul `MathTrans` nicht verwendet werden soll. Es entspricht in der Funktionalität dem Modul `MathLib0`, verwendet jedoch für alle Argumente den Typ FFP.

```
DEFINITION MODULE MathLibFFP;

FROM SYSTEM IMPORT
 FFP;

CONST
 e=2.7182818284;
 pi=3.1415926536;

PROCEDURE sqrt(x:FFP):FFP;
PROCEDURE exp(x:FFP):FFP;
PROCEDURE ln(x:FFP):FFP;
PROCEDURE sin(x:FFP):FFP;
PROCEDURE cos(x:FFP):FFP;
PROCEDURE arctan(x:FFP):FFP;

END MathLibFFP.
```

Die Einheit der Argumente für `sin` und `cos` sowie des Resultats von `arctan` ist Bogenmass (Radiant Abk. rad). Dabei gilt folgende Formel:

$$1 \text{ rad} = \frac{180°}{\pi}$$

`ln` gibt den natürlichen Logarithmus (Basis `e`) von `x` zurück.

Beispiel:

```
MODULE mlffp

FROM MathLibFFP IMPORT          (* !! *)
 e, pi,
 sqrt, exp, ln, sin, cos, arctan;

TYPE REAL = FFP;          (* !! *)

VAR
 r1, r2: REAL;

BEGIN
...
```

Anmerkung: Im obigen Beispiel müssen nur die beiden mit (* !! *) markierten Zeilen geändert werden, falls der Typ FFP nicht vorhanden ist.

MathLibLong

Standard Modul für einige mathematische Grundfunktionen. Diese Prozeduren arbeiten mit dem Typ LONGREAL.

```
DEFINITION MODULE MathLibLong;

CONST
 e=2.71828182845904523536;
 pi=3.14159265358979323846;

PROCEDURE sqrt(x:LONGREAL):LONGREAL;
PROCEDURE exp(x: LONGREAL):LONGREAL;
PROCEDURE ln(x:LONGREAL):LONGREAL;
PROCEDURE sin(x:LONGREAL):LONGREAL;
PROCEDURE cos(x:LONGREAL):LONGREAL;
PROCEDURE arctan(x:LONGREAL):LONGREAL;

END MathLibLong.
```

Die Einheit der Argumente für `sin` und `cos` sowie des Resultats von `arctan` ist Bogenmass (Radiant Abk. rad). Dabei gilt folgende Formel:

$$1\ \text{rad} = \frac{180°}{\pi}$$

`ln` gibt den natürlichen Logarithmus (Basis `e`) von `x` zurück.

Warnung:	Funktionsrufe mit Argumenten, deren Resultatwert nicht definiert ist (beispielsweise Quadratwurzel oder Logarithmus einer negativen Zahl), führen zum Abbruch des Programms.

RandomNumber

Dieses Modul generiert Pseudo-Zufallszahlen, wie sie in Spielprogrammen oder Simulationen benötigt werden. Dabei wird immer von einem Ausgangswert aus eine Zufallszahl sowie ein neuer Ausgangswert erzeugt.

```
DEFINITION MODULE RandomNumber;

PROCEDURE Random():REAL;

PROCEDURE RND(nr: INTEGER):INTEGER;

PROCEDURE GetSeed(VAR seed: LONGINT);

PROCEDURE PutSeed(seed: LONGINT);

END RandomNumber.
```

`Random` gibt eine Pseudo-Zufallszahl im Bereich von 0 bis 1 ([0,1[) zurück. Die erzeugten Zahlen sind gleichmässig über diesen Bereich verteilt. Mit dem vorgegebenen Startwert beträgt die Periode mindestens 2^{29}.

Die Funktion `RND` gibt eine ganze Zahl zwischen 0 und `nr-1` (beziehungsweise `nr+1` für negatives `nr`) zurück. Ein Würfel würde folgendermassen simuliert:

```
wurf:=1+RND(6);
```

`GetSeed` weist dem Parameter `seed` den aktuellen Ausgangswert zu.

`PutSeed` setzt einen neuen Ausgangswert. Dieser muss positiv sein.

RealConversions

Das Modul `RealConversions` stellt die Prozeduren zur Umwandlung von Gleitpunkt-Zahlen (Dezimalbrüche) zwischen interner und lesbarer Darstellung und umgekehrt zur Verfügung.

```
DEFINITION MODULE RealConversions;

PROCEDURE StrToReal(VAR s: ARRAY OF CHAR; VAR r: REAL;
                    VAR err: BOOLEAN);

PROCEDURE RealToStr(r: REAL; VAR s: ARRAY OF CHAR;
                    m, n: INTEGER; expo: BOOLEAN;
                    VAR err: BOOLEAN);

END RealConversions.
```

Die Prozedur `StrToReal` wandelt die in `str` gespeicherte Gleitpunktzahl in die interne REAL-Darstellung um. Enthält die Zeichenkette nicht erlaubte Zeichen, oder kann die beschriebene Zahl nicht in einer REAL-Variablen dargestellt werden, erhält `err` den Wert TRUE.

Die Prozedur `RealToStr` erlaubt vielseitige Umwandlungen von `r` in lesbare Darstellungsformen. Die resultierende Zeichenkette enthält genau `m` Zeichen, oder soviele wie `s` Platz bietet, falls `m` zu gross ist. Ist `m` positiv, werden notwendige Leerzeichen vor der Zahl angefügt, sonst werden sie angehängt. Die Anzahl Zeichen hinter dem Dezimalpunkt wird durch `n` gegeben. Falls in Berücksichtigung zur Feldgrösse die Anzahl der Dezimalstellen zu gross gewählt wurde, wird diese auf das entsprechende Maximum gekürzt. Ist `n=0` wird kein Dezimalpunkt geschrieben. Der Parameter `expo` gibt an, ob die wissenschaftliche Schreibweise (Exponentialdarstellung) gewünscht ist.

Beispiele:

```
r:=12.976
RealToStr(r,str,5,2,FALSE,err);
  (* str: "12.98"
     err: FALSE *)
RealToStr(r,str,4,2,FALSE,err);
  (* str: "13.0"
     err: FALSE *)
RealToStr(r,str,3,2,FALSE,err);
  (* str: " 13"
     err: FALSE *)
RealToStr(r,str,2,2,FALSE,err);
  (* str: "13"
     err: FALSE *)
RealToStr(r,str,1,2,FALSE,err);
  (* str: undefiniert
     err: TRUE *)
RealToStr(r,str,-8,2,FALSE,err);
  (* str: "12.98    "
     err: FALSE *)
RealToStr(r,str,10,2,TRUE,err);
  (* str: "  1.30E+01"
     err: FALSE *)
```

Mögliche Fehlerquellen (`err=TRUE`) sind ein negativer Wert von `n`, der Wert Null für `m` oder dass `s` zuwenig Zeichen beinhaltet, um die resultierende Zeichenkette abzuspeichern.

RealInOut

Standardmodul für die Ein- und Ausgabe von Zahlen des Typs REAL. Der Ein- und Ausgabestrom wird vom Modul `InOut` kontrolliert.

```
DEFINITION MODULE RealInOut;

VAR
 done:BOOLEAN;

PROCEDURE WriteReal(r: REAL; m,n: INTEGER);

PROCEDURE ReadReal(VAR r: REAL);

END RealInOut;
```

Die Prozedur `WriteReal` schreibt die übergebene Zahl `r` in ein Feld der Grösse `m`, wobei im Maximum `n` Dezimalstellen geschrieben werden. Füllt die Darstellung das Feld nicht aus, wird das Feld mit Leerzeichen ausgefüllt. Ist die Feldgrösse positiv, werden die Leerzeichen vor die Zahl gesetzt, sonst werden sie angehängt.

`ReadReal` liest eine Zeichenkette von der Eingabe (InOut) und wandelt sie in die REAL-Zahl `r` um.

Scan

```
DEFINITION MODULE Scan;

TYPE
 ReadProc=PROCEDURE(VAR CHAR);

PROCEDURE ScanString(read: ReadProc;
                     VAR str: ARRAY OF CHAR;
                     VAR len: INTEGER;
                     VAR termCh: CHAR);

END Scan.
```

Die Prozedur `ScanString` liest eine Zeichenkette ein und speichert diese in `str`. Führende Leerzeichen werden übersprungen. Die Leseprozedur muss dazu übergeben werden. Der Lesevorgang der Zeichenkette wird durch ein Leerzeichen oder *jedes* Kontrollzeichen (ch ≤ " ") beendet. Soll ein Leerzeichen in die Zeichenkette eingefügt werden, muss dieses in Anführungszeichen (") gesetzt werden. Das letztgelesene Zeichen wird in `termCh` gespeichert, die Länge der Zeichenkette in `len`.

Beispiele:

```
Dieser Text ...
    "Dieser" wird zurückgegeben.
Dies"mal ist´s" besser
    "Diesmal ist´s" wird zurückgegeben.
```

Storage

Dieses Standardmodul stellt die zur dynamischen Speicherverwaltung notwendigen Prozeduren zur Verfügung. Es behält die Kontrolle über alle allozierten Speicherbereiche, und gibt sie im Falle eines Programmabsturzes oder am Programmende wieder frei.

```
DEFINTION MODULE Storage;

FROM SYSTEM IMPORT
 ADDRESS;

PROCEDURE ALLOCATE(VAR adr: ADDRESS; size: LONGINT);
PROCEDURE DEALLOCATE(VAR adr: ADDRESS; size: LONGINT);
PROCEDURE Available(size: LONGINT): BOOLEAN;

END Storage.
```

Prozedur ALLOCATE alloziert einen Bereich der in Byte gegebenen Grösse und weist dessen Adresse dem Parameter adr zu. Ist nicht genügend Speicher vorhanden, bricht das aufrufende Programm mit einem Laufzeitfehler ab.

DEALLOCATE gibt einen zuvor allozierten Bereich an der Adresse adr wieder frei. Der übergebene Zeiger erhält den Wert NIL. Um mit gebräuchlichen Versionen kompatibel zu bleiben, muss die Grösse übergeben werden, der Wert wird allerdings ignoriert.

Die Funktion Available gibt den Wert TRUE zurück, falls ein Speicherbereich der gewünschten Grösse zur Verfügung steht.

Anmerkung: Amiga Modula-2 unterstützt die automatische Umsetzung von NEW und DISPOSE in ALLOCATE und DEALLOCATE nicht mehr. ALLOCATE und DEALLOCATE müssen explizit aufgerufen werden.

Str

Das Modul Str bietet einfache Manipulationsprozeduren für Zeichenketten an. Entgegen unserer Philosophie, (fast) alles in Modula-2 zu programmieren, ist dieses Modul in Assembler implementiert. Für den Benutzer spielt dies jedoch keine Rolle, die Anwendung ist durch die Modula-2-Definition gegeben.

```
DEFINITION MODULE Str;

CONST
 first=0;
 noOccur=-1;
 last=-1;

PROCEDURE Length(str: ARRAY OF CHAR): LONGCARD;
PROCEDURE CopyPos(VAR dest: ARRAY OF CHAR;
                  src: ARRAY OF CHAR;
                  destPos: LONGCARD);
PROCEDURE Copy(VAR dest: ARRAY OF CHAR;
               src: ARRAY OF CHAR);
PROCEDURE Concat(VAR dest: ARRAY OF CHAR;
                 src: ARRAY OF CHAR);
PROCEDURE Compare(str1, str2: ARRAY OF CHAR): LONGINT;
PROCEDURE FirstPos(str: ARRAY OF CHAR; from: LONGINT;
                   ch: CHAR): LONGINT;
PROCEDURE LastPos(str: ARRAY OF CHAR; to: LONGINT;
                  ch: CHAR): LONGINT;
PROCEDURE CapString(VAR str: ARRAY OF CHAR);

END Str.
```

Dieses Modul stellt die wichtigsten Prozeduren zur Manipulation von Zeichenketten zur Verfügung. Alle Routinen wurden in Assembler programmiert, um möglichst kompakt und schnell zu sein. Die Kompaktheit einer solchen Implementation ist aus der Grösse der Objektdatei leicht ersichtlich. Bei der Auswahl wurden nur häufig gebrauchte, einfache Prozeduren berücksichtigt.

Die Prozeduren verwenden wenn möglich Wertparameter. Dies ermöglicht die Übergabe konstanter Zeichenketten, was jedoch nicht immer effizient ist:

Beispiel:
```
CONST
 halloWelt = "Hallo Welt"

Length(halloWelt) = SIZE(halloWelt)
```

Entgegen der normalen Behandlung von Wertparametern werden von den übergebenen Zeichenketten *keine* lokalen Kopien erstellt. Dadurch erreicht dieses Modul dieselbe Effizienz, die mit Variablen-Parametern erreicht werden kann. Bei der Prozedur `CopyPos` kan dies aber zu Problemen führen, falls innerhalb der Zeichenkette umkopiert werden soll.

Die Funktion `Length` gibt die Anzahl Zeichen in `str` zurück. Ein abschliessendes Nullzeichen wird nicht gezählt. Es gilt also immer folgenden Ungleichung:

```
0 ≤ Length(string) ≤ HIGH(string) + 1
```

Die Prozedur `CopyPos` kopiert die Zeichen aus `src` in die Zeichenreihe `dest`, wobei an der Stelle `dest[destPos]` begonnen wird. Die nachfolgenden Zeichen in `src` werden überschrieben. Ist die resultierende Zeichenkette zu lang, wird der Rest abgeschnitten.

Die Prozedur `Copy` überschreibt die Zeichenkette `dest` mit dem Inhalt von `src`. Es entspricht einem Aufruf von `CopyPos`, wobei `destPos` zu `0` gesetzt wird.

`Concat` hängt die Zeichenkette `src` an `dest` an. Überzählige Zeichen werden abgeschnitten.

`Compare` vergleicht die Zeichenkette `str1` mit der Zeichenkette `str2`. Ist das Resultat negativ, so ist `str2` kleiner, ist es Null, sind beide gleich, und ist es positiv, so ist `str2` grösser als `str1`. Der Absolutwert des zurückgegebenen Resultats gibt die Zeichenposition, an der sich die beiden Zeichenketten unterscheiden, wobei das erste Zeichen dem Wert eins entspricht.

Die Funktion `FirstPos` sucht nach dem ersten Vorkommen des Zeichens `ch` innerhalb der Zeichenkette `str`, dort startend bei der Position `from`. Zurückgegeben wird die Position des ersten Vorkommens, oder `noOccur` falls das Zeichen nicht gefunden wurde.

Die Funktion `LastPos` sucht nach dem letzten Vorkommen des Zeichens `ch` innerhalb der Zeichenkette `str` .Gesucht wird bis zur Position `to`. Zurückgegeben wird die Position des letzten Vorkommens, oder `noOccur` falls das Zeichen nicht gefunden wurde.

`CapString` wandelt alle Kleinbuchstaben der Zeichenkette `str` in die entsprechenden Grossbuchstaben um. Die Umwandlung entspricht derjenigen der Funktion `CAP()`, wird also nur für die Zeichen `"a"-"z"` ausgeführt.

Strings

Das Modul Strings bietet einige Manipulationsprozeduren für Zeichenketten an.

```
DEFINITION MODULE Strings;

CONST
 first=0;
 last=-1;

PROCEDURE Length(VAR str: ARRAY OF CHAR): INTEGER;
PROCEDURE Occurs(VAR str: ARRAY OF CHAR; from:INTEGER;
                 token: ARRAY OF CHAR;
                 caseSens: BOOLEAN): INTEGER;
PROCEDURE Insert(VAR str: ARRAY OF CHAR; at: INTEGER;
                 token:ARRAY OF CHAR);
PROCEDURE Delete(VAR str: ARRAY OF CHAR;
                 start, length: INTEGER);
PROCEDURE Copy(VAR str: ARRAY OF CHAR;
               source: ARRAY OF CHAR;
               start, length: INTEGER);
PROCEDURE Compare(VAR str: ARRAY OF CHAR;
                  start, length: INTEGER;
                  token: ARRAY OF CHAR;
                  caseSens: BOOLEAN):INTEGER;

END Strings.
```

Die Funktion `Length` gibt die Anzahl Zeichen in `str` zurück. Ein abschliessendes Nullzeichen wird nicht gezählt.

Die Funktion `Occurs` sucht nach dem Vorkommen von `token` in `str`, wobei dort mit dem Zeichen an der Position `from` begonnen wird. Der Resultatwert entspricht der Position des ersten gemeinsamen Zeichens, oder `last` falls `token` in `str` nicht vorkommt. Ist der Parameter `caseSens` wahr, werden Gross- und Kleinbuchstaben als unterschiedlich betrachtet, sonst als gleichbedeutend. Davon ausgenommen sind alle Zeichen mit einem Ordinalwert grösser 127.

Die Prozedur `Insert` fügt die Zeichenkette `token` links des Zeichens an der Position `at` in `str` ein. Ist die resultierende Zeichenkette zu lang,

um abgelegt zu werden, wird diese entsprechend abgeschnitten. Falls notwendig, d.h. falls `at > Length(str)`, werden der Zeichenkette `str` zuerst Leerzeichen angehängt. Ist `at = last`, wird `token` an `str` angehängt.

`Delete` löscht `length` Zeichen aus `str` heraus, beginnend an der Stelle `from`. Die dahinterliegenden Zeichen rutschen nach.

`Copy` kopiert eine Teilzeichenkette von `source` nach `str`. Die Teilzeichenkette beginnt an der Stelle `start` und ist `lentgh` Zeichen lang. Zeichen, die hinter dem letzten Zeichen von `source` liegen, werden nicht berücksichtigt. Die Zielzeichenkette wird analog zu `Insert` abgeschnitten, falls der Platz nicht ausreichen sollte.

`Compare` vergleicht die Teilzeichenkette von `str`, beginnend an der Stelle `start` und `length` Zeichen lang, mit der gegebenen Zeichenkette `token`. Ist das Resultat negativ, so ist `token` kleiner, ist es Null, sind beide gleich, und ist es positiv, so ist `token` grösser als die Teilzeichenkette. Der Absolutwert des zurückgegebenen Resultats gibt die Zeichenposition, an der sich die beiden Zeichenketten unterscheiden, wobei das erste Zeichen dem Wert eins entspricht.

Anmerkung: Der Parameter `str` in `Length`, `Occurs` und `Compare` ist aus Effizienzgründen ein Variablen-Parameter. Die Zeichenkette wird nicht verändert. Für konstante Zeichenketten sind diese Funktionen sinnlos, da die Resultate sowieso bekannt sind.

SYSTEM

Anmerkung: Dies kann nur der Versuch eines Definitionsmoduls sein. Für Einzelheiten sei auf den Teil der Compiler-Beschreibung verwiesen.

```
DEFINITION MODULE SYSTEM;
TYPE
 BYTE;
 WORD;
 ADDRESS = POINTER TO BYTE;
 BPTR;
 BITSET = SET OF [0..15];
 LONGSET = SET OF [0..31];
 FFP;

(*
 * T bezeichnet skalaren Typ,
 * dessen Grösse <= 4 Byte ist
 *)

PROCEDURE ADR    (VAR x: AnyType): ADDRESS;
PROCEDURE INLINE(x : WORD);
PROCEDURE REG    (reg: [0..15]): LONGINT;
PROCEDURE SETREG(reg: [0..15]; val: T);
            (* reg const *)
            (* SIZE(T)=4 *)
PROCEDURE TSIZE  (AnyType): LONGINT;
PROCEDURE SHIFT  (x: T; n: LONGINT): T;
PROCEDURE CAST   (AnyType1; x: AnyType2): AnyType1;
            (* SIZE(AnyType1) = SIZE(AnyType2) *)

END SYSTEM.
```

Terminal

Das Standard Modul `Terminal` bietet einige grundlegende Prozeduren um Zeichen von der Tastatur zu lesen und auf den Bildschirm zu schreiben.

```
DEFINITION MODULE Terminal;

VAR
 waitCloseGadget: BOOLEAN;

PROCEDURE Read(VAR ch: CHAR);
PROCEDURE ReadLn(VAR str: ARRAY OF CHAR;
                 VAR len: INTEGER);
PROCEDURE BusyRead(VAR ch: CHAR);

PROCEDURE Write(ch: CHAR);
PROCEDURE WriteLn;
PROCEDURE WriteString(string: ARRAY OF CHAR);

END Terminal.
```

Das Modul `Terminal` berücksichtigt die Unterschiede, die sich aus der Arbeitsumgebung ergeben. Wird das Programm, welches das Modul `Terminal` importiert, vom CLI aus aufgerufen, läuft die Ein- und Ausgabe über das entsprechende CLI-Fenster (Dos.Input und Dos.Output). Wird dasselbe Programm von der Workbench aus aufgerufen, wird ein Fenster geöffnet, über das die Ein- und Ausgabe laufen kann.

Die Prozedur `ReadLn` liest eine ganze Zeile vom Eingabestrom, d.h. bis das Zeilenende (`ASCII.eol`) oder das Eingabeende (`ASCII.eof`) erreicht wird. Die eingelesenen Zeichen werden in `str` abgelegt. Ist in der Variablen `str` nicht genügend Platz vorhanden, wird der überschüssige Teil ignoriert und ist für das Programm verloren. Dürfen keine Zeichen der Eingabe verloren gehen, muss die Prozedur Read verwendet werden.

Prozedur `Read` liest ein einzelnes Zeichen ein. Am Ende der Eingabedatei wird das Zeichen `ASCII.eof` zurückgegeben. Dies ist nötig, um das Eingabeende bei Eingabeumlenkung zu erkennen. Man

beachte, dass von einem Console-Fenster die Eingabe erst empfangen werden kann, wenn die Zeile mit <RETURN> abgeschlossen wurde.

In einem Console-Fenster kann `ASCII.eol` durch <CTRL-\> an das Programm übergeben werden. Dies ist sinnvoll, um das Verhalten bei Dateiende zu testen oder zu simulieren.

Die Prozedur `BusyRead` ist aus Gründen der Kompatibilität zur ursprünglichen Definition von Terminal (→ [PIM3]) übernommen worden. Der Grundgedanke von `BusyRead` ist die Abfrage, ob im Eingabestrom Zeichen zur Übernahme bereit stehen. In der üblichen Umgebung eines Console-Fensters ist dies aber nicht möglich. Diese Implementation von `BusyRead` liest das nächste Zeichen von der Eingabe und weist das gelesene Zeichen der Variablen `ch` zu. Ist kein Zeichen im Eingabestrom vorhanden, erhält `ch` den Wert `ASCII.nul`. So ist es *nicht* möglich, das Dateiende einer interaktiven Datei (Eingabe von <CTRL-\>) zu erkennen. Falls die Eingabe mittels Eingabeumlenkung von einem nicht-interaktiven Eingabestrom kommt, ist `BusyRead` identisch mit `Read`.

Mit `Write` wird ein einzelnes Zeichen zur aktuellen Ausgabe ausgegeben. Aus Geschwindigkeitsgründen sollte man die Ausgabe einzelner Zeichen vermeiden und wenn immer möglich `WriteString` verwenden.

`WriteLn` setzt den Cursor auf die erste Spalte der folgenden Zeile.

Die Prozedur `WriteString` gibt die übergebene Zeichenkette aus. Die Zeichenkette kann durch das Zeichen `ASCII.nul` abgeschlossen sein.

Anmerkung: Ein laufendes Programm kann durch Drücken von ^C im (aktivierten) Terminal-Fenster unterbrochen werden. Unter Umständen muss mehrmals ^C gedrückt werden.

Verwendeter Ein- und Ausgabekanal

Im Normalfall ist das von Terminal verwendete Fenster ein Console-Fenster (Aufruf vom CLI oder von der Workbench). Dadurch ergeben sich Unterschiede zu manchen Modula-2-Umgebungen. Während es dort möglich ist, eine Eingabe von der Tastatur unmittelbar zu lesen, ist dies in einem Console-Fenster nicht möglich. Ein Console-Fenster erlaubt die Änderung der gesamten Eingabezeile solange, bis die Eingabe mit <RETURN> bestätigt wird. Ein einzelnes Zeichen kann also nicht eingelesen werden, bevor der Anwender die <RETURN>-Taste betätigt hat.

Beispiel:
```
PROCEDURE Ja(text: ARRAY OF CHAR): BOOLEAN;
(* TRUE, falls Benutzer mit "j" bestätigt *)
VAR
 ch: CHAR;
BEGIN
 WriteString(text);
 Read(ch);
 RETURN CAP(ch)="J"
END Ja;
```

In diesem Beispiel bleiben einige Zeichen im Eingabestrom zurück, mindestens das Zeilenendzeichen (ASCII.eol). Eine bessere (weil sicherere) Variante dieser Funktion sähe folgendermassen aus:

Beispiel:
```
PROCEDURE Ja(text: ARRAY OF CHAR): BOOLEAN;
(* TRUE, falls Benutzer mit "j" bestätigt *)
VAR
 l: INTEGER;
 str: ARRAY [0..3] OF CHAR;
BEGIN
 WriteString(text);
 ReadLn(str,l);
 RETURN CAP(str[0])="J"
END Ja;
```

Diese Funktion liest eine ganze Zeile ein. Der Benutzer kann mit einem einfachen <RETURN> die Frage verneinen, oder mit "j"<RETURN> die Frage bejahen. Jede Zeichenkette, die mit "j" oder "J" beginnt und mit

<RETURN> abgeschlossen wird, gilt als Zustimmung. "Jein" würde somit als "Ja" interpretiert.

Ist das Programm von der Workbench aus gestartet, wird ein Console-Fenster mit obigen Eigenschaften eröffnet. Die Grösse dieses Fensters ist im Modul Terminal vorgegeben. die Vorgabe ist `CON:0/50/640/100/`. Es besteht jedoch die Möglichkeit, Art und Grösse des Fensters selbst zu bestimmen. Vor dem Öffnen des Fensters mit den Vorgabewerten wird in der Ikone des Programms nach einem Eintrag "WINDOW=" gesucht. Dieser Eintrag muss ein gültiger Fenster-Gerätename sein (→ [DOS]). Der Fenstername dieser Spezifikation wird ignoriert; Terminal setzt stattdessen den aktuellen Programmnamen ein.

Beispiel: Eintrag in Tooltypes:
```
WINDOW=CON:0/0/640/200/
```

Anmerkung: Für besondere Anwendungen ist es auch möglich, andere Ein- und Ausgabekanäle zu verwenden. Dabei ist zu beachten, dass Ein- und Ausgabe über den gleichen Kanal abgewickelt werden.

```
WINDOW=AUX:
WINDOW=SER:                    usw.
```

Wird für den Workbench-Aufruf ein Fenster geöffnet, wird dieses mit einem Closing-Gadget ausgestattet. Das Fenster bleibt nach Programmende offen, bis der Benutzer das Closing-Gadget anklickt. So bleibt die Programmausgabe sichtbar. Soll das Fenster unmittelbar nach Programmende geschlossen werden, muss die Variable `waitCloseGadget` auf `FALSE` gesetzt werden. Der Vorgabewert dieser Variablen ist `TRUE`.

Falls beim Aufruf von der Workbench beim Eröffnen des Terminal-Fensters ein Fehler auftritt, erscheint in einem Requester eine der beiden folgenden Meldungen:

Terminal: Open()-Fehler

Beim Erröffnen des Fensters wurde vom Dos ein Fehler zurückgemeldet. Möglicherweise konnte das Fenster aus Speichermangel nicht geöffnet werden.

Terminal: WINDOW=???

Die Fensterspezifikation ist nicht korrekt oder ungültig. Kontrollieren Sie vor allem, ob das Fenster überhaupt im "Screen" Platz hat.

Anmerkung: Diese Implementation von Terminal ist für den allgemeinen Fall, die Verwendung mit Console-Fenstern und sequentiellen Dateien bei der Ein- und Ausgabeumlenkung konzipiert. Darum ist es möglich, dass unter bestimmten Umständen (z.B. RAW: als Ein-/Ausgabekanal) nicht alle Funktionen exakt der Beschreibung folgen.

Windows

Das Modul Windows bietet eine einfache Schnittstelle, um mit Fenstern zu arbeiten. Für die einfache Handhabung wird keine detaillierte Kenntnis von Intuition gefordert.

```
DEFINTION MODULE Windows;

IMPORT Intuition;

TYPE
 Window=Intuition.WindowPtr;
 WinGad=(sizing,moving,arranging,closing,scrolling);
 WinGadSet=SET OF WinGad;
 Mode=(replMd,xorMd,invMd);
 ModeSet=SET OF Mode;
 (* ModeSet{}=orMd *)

PROCEDURE OpenWindow(VAR u: Window; x,y,w,h: INTEGER;
                     title: ARRAY OF CHAR;
                     gad:WinGadSet);
PROCEDURE CloseWindow(VAR u:Window);
PROCEDURE GetSize(VAR u: Window;
                  VAR line,column: INTEGER);
PROCEDURE XYtoPos(VAR u:Window; x,y: INTEGER;
                  VAR line,column: INTEGER);
PROCEDURE SetPos(VAR u: Window; line,column: INTEGER);
PROCEDURE GetPos(VAR u: Window;
                 VAR line,column: INTEGER);
PROCEDURE SetColor(VAR u: Window; fg,bg: INTEGER);
PROCEDURE SetMode(VAR u: Window; mode: ModeSet);
PROCEDURE Scroll(VAR u: Window; nr: INTEGER);
PROCEDURE Clear(VAR u: Window);
PROCEDURE WriteC(VAR u: Window; ch: CHAR);
PROCEDURE WriteS(VAR u: Window; str: ARRAY OF CHAR);
PROCEDURE WriteL(VAR u: Window);
(* fortgeschrittene Anwendung *)
PROCEDURE SetScreen(scr: Intuition.ScreenPtr);
PROCEDURE SetClip(VAR u: Window; on: BOOLEAN);

END Windows.
```

`OpenWindow` öffnet ein Fenster mit den angegebenen Koordinaten `x`, `y`, `w`, `h` (in Bildpunkten, Nullpunkt links oben) und der Kopfzeile `title`.

Alle in Intuition verfügbaren Standard-Gadgets sowie ein Rollbalken auf der rechten Seite können dem Fenster zugeordnet werden. So zugewiesene Gadgets müssen vom Benutzer behandelt werden. Ohne spezielle Behandlung sollten nur das `arranging`- und das `moving`-Gadget zugewiesen werden.

Als Schreibmodus wird `replMd` gesetzt und die Cursorposition ist 0,0 (links oben). Konnte das Fenster nicht geöffnet werden, erhält `u` den Wert `NIL`.

`CloseWindow` schliesst das durch `u` identifizerte Fenster wieder.

Die Prozedur `GetSize` liefert die Zeilen- und Spaltenzahl des Fensters `u`.

`XYtoPos` setzt die `x`- und `y`-Koordinate relativ zum Fensternullpunkt in die Fensterzeile `l` und -spalte `c` um.

Der unsichtbare Cursor kann mit `SetPos` gesetzt werden, während `GetPos` die aktuelle Cursorposition liefert (immer in Zeile und Spalte gerechnet).

Beim Eröffnen wird die Hinter- und Vordergrundfarbe auf 0 und 1 gesetzt. `SetColor` lässt diese Farbwerte ändern. Soll nur ein Wert verändert werden, kann dem anderen -1 übergeben werden.

`SetMode` setzt den Schreibmodus für ein bestimmtes Fenster. Mit `replMd` wird das Buchstabenmuster über den momentanen Inhalt gelegt, `xorMd` ändert die Farbe an den im Muster gesetzten Stellen und `orMd` setzt die Farbe an den im Muster gesetzten Stellen. Ist `invMd` gesetzt, wird das Buchstabenmuster zuerst invertiert, bevor einer der drei beschriebenen Modi zur Anwendung kommt.

`Scroll` bewegt den Fensterinhalt um soviele Zeilen wie angegeben, aufwärts bei positiven Grössen und abwärts bei negativem `nr`.

Ein Fenster kann durch die Prozedur `Clear` gelöscht werden. Das Fenster wird mit der Hintergrundfarbe ausgefüllt.

`WriteC` schreibt an der aktuellen Cursorposition ein Zeichen, und bewegt den Cursor entsprechend.

`WriteS` schreibt die übergebene Zeichenkette an der aktuellen Cursorposition. Geht die Zeichenkette über den rechten Rand hinaus, wird auf der nächsten Zeile weitergefahren (Ausnahme siehe unter `SetClip`).

`WriteL` setzt den Cursor an der Anfang der nächsten Zeile.

Die Fenster werden normalerweise auf dem Workbench-Bildschirm eröffnet. Mit `SetScreen` kann jedoch für alle nachfolgenden Aufrufe von `Openwindow` ein anderer Bildschirm gesetzt werden. Um wieder zum Workbench-Bildschirm zurückkehren zu können, muss `NIL` übergeben werden.

Mit der Prozedur `SetClip` kann angegeben werden, ob ein zu schreibender Text, der nicht vollständig auf der Fensterzeile Platz hat, abgeschnitten oder auf der nächsten Zeile weitergeschrieben werden soll. Wird der Clip-Modus gesetzt (`on=TRUE`), bewirkt ausserdem ein Zeilenvorschub am Ende des Fensters bloss das Setzen des Cursors auf das erste Zeichen der letzten Zeile.

Anmerkung: Das Window hat noch keine IDCMPFlags und somit auch keinen Replyport. Mit der Intuition Prozedur ModifyIDCMP kann dem Window ein Port zugewiesen werden.

13 Schnittstellen-Moduln

Audio 3

BootBlock 4

Clipboard 5

Console 6

ConUnit 8

DiskFont 10

Dos 12

Exec 19

ExecSupport 29

Expansion 30

GamePort 34

GfxMacros 35

Graphics 36

Hardware 50

Icon 55

Input 56

InputEvent 57

Intuition 59

Keyboard 77

KeyMap 78

Layers 80

MathIEEEDoubTrans 82

MathTrans 83

Narrator 84

Parallel 86

Printer 87

PrtBase 90

Resources 92

Serial 95

Timer 97

TrackDisk 98

Translator 100

Workbench 101

Audio

```
 5 (*$M-*)
 6 DEFINITION MODULE Audio;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Exec IMPORT
11  nonstd,IOFlagSet,IORequest,Message;
12
13 CONST
14  audioName="audio.device";
15  hardChannels=4;
16  allocMinprec=-128;
17  allocMaxprec=127;
18  free=nonstd+0;
19  setPrec=nonstd+1;
20  finish=nonstd+2;
21  perVol=nonstd+3;
22  lock=nonstd+4;
23  waitCycle=nonstd+5;
24  noUnit=32;
25  allocate=noUnit+0;
26  pervol=IOFlagSet{4};
27  syncCycle=IOFlagSet{5};
28  noWait=IOFlagSet{6};
29  writeMessage=IOFlagSet{7};
30  noAllocation=-10;
31  allocFailed=-11;
32  channelStolen=-12;
33
34 TYPE
35  IOAudio=RECORD
36   request:IORequest;
37   allocKey:INTEGER;
38   data:ADDRESS;
39   length:LONGCARD;
40   period:CARDINAL;
41   volume:CARDINAL;
42   cycles:CARDINAL;
43   writeMsg:Message;
44  END;
45  IOAudioPtr=POINTER TO IOAudio;
46
47 END Audio.
48
```

BootBlock

```
(* $M- *)
DEFINITION MODULE BootBlock;

FROM SYSTEM IMPORT CAST;

TYPE
 BootBlock=RECORD
  id:ARRAY [0..3] OF CHAR;
  chkSum:LONGINT;
  dosBlock:LONGINT;
 END;

CONST
 bootSects=2;
 idDos=CAST(LONGINT,'DOS ')-ORD(' ');
 idKick=CAST(LONGINT,'KICK');
 nameDos='DOS';
 nameKick='KICK';

END BootBlock.

```

Clipboard

```
(* $M- *)
DEFINITION MODULE Clipboard;

FROM SYSTEM IMPORT
 ADDRESS;
FROM Exec IMPORT
 nonstd,Node,Message,DevicePtr,IOFlagSet,UnitPtr;

CONST
 clipboardName="clipboard.device";
 post=nonstd+0;
 currentReadId=nonstd+1;
 currentWriteId=nonstd+2;
 obsoleteId=1;

TYPE
 ClipboardUnitPartial=RECORD
  node:Node;
  unitNum:LONGCARD;
 END;
 ClipboardUnitPartialPtr=POINTER TO ClipboardUnit;
 IOClipboard=RECORD
  message:Message;
  device:DevicePtr;
  unit:UnitPtr;
  command:CARDINAL;
  flags:IOFlagSet;
  error:[-128..127];
  actual:LONGCARD;
  length:LONGCARD;
  data:ADDRESS;
  offset:LONGCARD;
  clipID:LONGINT;
 END;
 IOClipboardPtr=POINTER TO IOClipReqPtr;

CONST
 primaryClip=0;

TYPE
 SatisfyMsg=RECORD
  msg:Message;
  unit:CARDINAL;
  clipID:LONGINT;
 END;
 SatisfyMsgPtr=POINTER TO SatisfyMsg;

END Clipboard.

```

Console

```
 5 (*$M-*)
 6 DEFINITION MODULE Console;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Exec IMPORT
11  nonstd;
12 FROM InputEvent IMPORT
13  InputEventPtr;
14
15 CONST
16  consoleName="console.device";
17  askKeyMap=nonstd+0;
18  setKeyMap=nonstd+1;
19  askDefaultKeyMap=nonstd+2;
20  setDefaultKeyMap=nonstd+3;
21  primary=0;
22  bold=1;
23  italic=3;
24  underscore=4;
25  negative=7;
26  black=30;
27  red=31;
28  green=32;
29  yellow=33;
30  blue=34;
31  magenta=35;
32  cyan=36;
33  white=37;
34  default=39;
35  blackBg=40;
36  redBg=41;
37  greenBg=42;
38  yellowBg=43;
39  blueBg=44;
40  magentaBg=45;
41  cyanBg=46;
42  whiteBg=47;
43  defaultBg=49;
44  clr0=30;
45  clr1=31;
46  clr2=32;
47  clr3=33;
48  clr4=34;
49  clr5=35;
50  clr6=36;
51  clr7=37;
52  clr0Bg=40;
53  clr1Bg=41;
54  clr2Bg=42;
55  clr3Bg=43;
56  clr4Bg=44;
57  clr5Bg=45;
```

```
 1  clr6Bg=46;
 2  clr7Bg=47;
 3  dsrCpr=6;
 4  ctcHSetTab=0;
 5  ctcHClrTab=2;
 6  ctcHClrTabsAll=5;
 7  tbcHClrTab=0;
 8  tbcHClrTabsAll=3;
 9  mLnm=20;
10  mAsm=">1";
11  mAwm="?7";
12
13 PROCEDURE CDInputHandler(device0{14}:ADDRESS;
14                          events{8}:InputEventPtr;
15                          device1{9}:ADDRESS); CODE -42;
16 PROCEDURE RawKeyConvert(
17            device0{14}:ADDRESS;
18            events{8}:InputEventPtr;
19            buffer{9}:ADDRESS;
20            length{1}:LONGINT;
21            keyMap{10}:ADDRESS):LONGINT; CODE -48;
22
23 END Console.
24
```

ConUnit

```
 5 (* $M- *)
 6 DEFINITION MODULE ConUnit;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Console IMPORT
11  mLnm;
12 FROM Exec IMPORT
13  MsgPort,UByte;
14 FROM Graphics IMPORT
15  DrawModes,TextFontPtr;
16 FROM InputEvent IMPORT
17  classMax;
18 FROM Intuition IMPORT
19  WindowPtr;
20 FROM KeyMap IMPORT
21  KeyMap;
22
23 CONST
24  pmbAsm=mLnm+1;
25  pmbAwm=pmbAsm+1;
26  maxTabs=80;
27
28 TYPE
29  ConUnit=RECORD
30   mp:MsgPort;
31   window:WindowPtr;
32   xCP:INTEGER;
33   yCP:INTEGER;
34   xMax:INTEGER;
35   yMax:INTEGER;
36   xRSize:INTEGER;
37   yRSize:INTEGER;
38   xROrigin:INTEGER;
39   yROrigin:INTEGER;
40   xRExtant:INTEGER;
41   yRExtant:INTEGER;
42   xMinShrink:INTEGER;
43   yMinShrink:INTEGER;
44   xcCP:INTEGER;
45   ycCP:INTEGER;
46   keyMapStruct:KeyMap;
47   tabStops:ARRAY [0..maxTabs-1] OF CARDINAL;
48   mask:UByte;
49   fgPen:UByte;
50   bgPen:UByte;
51   aolPen:UByte;
52   drawMode:DrawModes;
53   areaPtSz:UByte;
54   areaPtrn:ADDRESS;
55   minTerms:ARRAY [0..7] OF UByte;
56   font:TextFontPtr;
57   algoStyle:UByte;
```

```
1    txFlags:UByte;
2    txHeight:CARDINAL;
3    txWidth:CARDINAL;
4    txBaseLine:CARDINAL;
5    txSpacing:CARDINAL;
6    modes:ARRAY [0..(pmbAwm+7) DIV 8-1] OF UByte;
7    rawEvents:ARRAY [0..(classMax+7) DIV 8-1] OF UByte;
8   END;
9   ConUnitPtr=POINTER TO ConUnit;
10
11 END ConUnit.
12
```

DiskFont

```
 5 DEFINITION MODULE DiskFont{"diskfont.library",33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS,BPTR;
 9 FROM Exec IMPORT
10  Node;
11 FROM Graphics IMPORT
12  FontFlagSet,FontStyleSet,TextAttr,TextAttrPtr,TextFont
13  ,TextFontPtr;
14
15 CONST
16  maxFontPath=256;
17  maxFontName=32;
18  fchId=0F00H;
19  dfhId=0F80H;
20
21 TYPE
22  AvailFontTypes=(
23   memory,disk,af2,af3,af4,af5,af6,af7,
24   af8,af9,af10,af11,af12,af13,af14,a15
25  );
26  AvailFontTypeSet=SET OF AvailFontTypes;
27  FontContents=RECORD
28   fileName: ARRAY [0..maxFontPath-1] OF CHAR;
29   ySize: CARDINAL;
30   style: FontStyleSet;
31   flags: FontFlagSet;
32  END;
33  FontContentsHeader=RECORD
34   fileId: CARDINAL;
35   numEntries: CARDINAL;
36 (*fc: ARRAY [0..numEntries-1] OF FontContents *)
37  END;
38  FontContentsHeaderPtr=POINTER TO FontContentsHeader;
39  DiskFontHeader=RECORD
40   df: Node;
41   fileId: CARDINAL;
42   revision: CARDINAL;
43   segment: BPTR;
44   name: ARRAY [0..maxFontName-1] OF CHAR;
45   tf: TextFont
46  END;
47  AvailFont=RECORD
48   type: AvailFontTypeSet;
49   attr: TextAttr;
50  END;
51  AvailFontHeader=RECORD
52   numEntries: CARDINAL;
53 (*af: ARRAY [0..numEntries-1] OF AvailFont;*)
54  END;
55  AvailFontHeaderPtr=POINTER TO AvailFontHeader;
56
57 PROCEDURE AvailFonts(
```

```
1             buffer{8}: ADDRESS;
2             bufBytes{0}: LONGINT;
3             types{1}: AvailFontSetType):LONGINT; CODE -36;
4  PROCEDURE OpenDiskFont(
5             textAttr{8}: TextAttrPtr): TextFontPtr; CODE -30;
6
7  END DiskFont.
8
```

Dos

```
5  DEFINITION MODULE Dos {"dos.library",33};
6
7  FROM SYSTEM IMPORT
8   ADDRESS,BPTR,BYTE,CAST,LONGSET,WORD;
9  FROM Exec IMPORT
10  Library,Message,MessagePtr,MsgPort,MsgPortPtr,Task;
11
12 CONST
13  dosName="dos.library";
14   (*
15    * Mögliche Werte des Parameters accessMode
16    * der Prozedur Open
17    *)
18  readWrite=1004;
19  readOnly=1005;
20  oldFile=readOnly;
21  newFile=1006;
22   (* Mögliche Werte des Parameters mode der Prozedur Seek *)
23  beginning=-1;
24  current=0;
25  end=1;
26   (*
27    * Mögliche Werte des Parameters accessMode
28    * der Prozedur Lock
29    *)
30  sharedLock=-2;
31  exclusiveLock=-1;
32  accessRead=sharedLock;      (* synonym *)
33  accessWrite=exclusiveLock; (* synonym *)
34
35  ticksPerSecond=50;
36
37 TYPE
38  BSTR=BPOINTER TO ARRAY [0..255] OF CHAR;
39  FileHandlePtr=BPOINTER TO FileHandle;
40  FileLockPtr=BPOINTER TO FileLock;
41  Date=RECORD
42   days:LONGINT;
43   minute:LONGINT;
44   tick:LONGINT;
45  END;
46  DatePtr=POINTER TO Date;
47  ProtectionFlags=(
48   delete,execute,writeProt,readProt,archive,script,pure,hidden,
49   pf8,pf9,pf10,pf11,pf12,pf13,pf14,pf15,pf16,pf17,pf18,pf19,pf20,
50   pf21,pf22,pf23,pf24,pf25,pf26,pf27,pf28,pf29,pf30,pf31
51  );
52  ProtectionFlagSet=SET OF ProtectionFlags;
53  FileInfoBlockPtr=POINTER TO FileInfoBlock;
54  FileInfoBlock=RECORD
55   diskKey:LONGINT;
56   dirEntryType:LONGINT;
57   fileName:ARRAY [0..107] OF CHAR;
```

```
  protection:ProtectionFlagSet;
  entryType:LONGINT;
  size:LONGINT;
  numBlocks:LONGINT;
  date:Date;
  comment:ARRAY [0..79] OF CHAR;
  padding:ARRAY [0..35] OF CHAR;
 END;
 InfoDataPtr=POINTER TO InfoData;
 InfoData=RECORD
  numSoftErrors:LONGINT;
  unitNumber:LONGINT;
  diskState:LONGINT;
  numBlocks:LONGINT;
  numBlocksUsed:LONGINT;
  bytesPerBlock:LONGINT;
  diskType:LONGINT;
  volumeNode:BPTR;
  inUse:LONGINT;
 END;

CONST
  (* Mögliche Werte von InfoData.diskState *)
 writeProtect=80;
 validating=81;
 validated=82;
  (* Mögliche Werte von InfoData.diskType *)
 noDiskPresent=-1;
 unreadableDisk=CAST(LONGINT,"BAD ")-ORD(" ");
 dosDisk=CAST(LONGINT,"DOS ")-ORD(" ");
 notReallyDos=CAST(LONGINT,"NDOS");
 kickstartDisk=CAST(LONGINT,"KICK");
  (* Mögliche Rückgabewerte von IoErr() *)
 noFreeStore=103;
 taskTableFull=105;
 lineTooLong=120;
 fileNotObject=121;
 invalidResidentLibrary=122;
 noDefaultDir=201;
 objectInUse=202;
 objectExists=203;
 dirNotFound=204;
 objectNotFound=205;
 badStreamName=206;
 objectTooLarge=207;
 actionNotKnown=209;
 invalidComponentName=210;
 invalidLock=211;
 objectWrongType=212;
 diskNotValidated=213;
 diskWriteProtected=214;
 renameAcrossDevices=215;
 directoryNotEmpty=216;
 tooManyLevels=217;
 deviceNotMounted=218;
 seekError=219;
 commentTooBig=220;
```

```
 1  diskFull=221;
 2  deleteProtected=222;
 3  writeProtected=223;
 4  readProtected=224;
 5  notADosDisk=225;
 6  noDisk=226;
 7  noMoreEntries=232;
 8   (*
 9    * Vordefinierte Werte für den Parameter returnCode
10    * der Prozedur Exit
11    *)
12  ok=0;
13  warn=5;
14  error=10;
15  fail=20;
16   (* break flags Exec.SetSignal, etc *)
17  ctrlC=12;
18  ctrlD=13;
19  ctrlE=14;
20  ctrlF=15;
21
22 TYPE
23  CommandLineInterfacePtr=BPOINTER TO CommandLineInterface;
24  ProcessId=MsgPortPtr;
25  ProcessPtr=POINTER TO Process;
26  Process=RECORD
27   task:Task;
28   msgPort:MsgPort;
29   pad:WORD;
30   segList:BPTR;
31   stackSize:LONGINT;
32   globVec:ADDRESS;
33   taskNum:LONGINT;
34   stackBase:BPTR;
35   result2:LONGINT;
36   currentDir:FileLockPtr;
37   cis:FileHandlePtr;
38   cos:FileHandlePtr;
39   consoleTask:ProcessId;
40   fileSystemTask:ProcessId;
41   cli:CommandLineInterfacePtr;
42   returnAddr:ADDRESS;
43   pktWait:ADDRESS;
44   windowPtr:ADDRESS;
45  END;
46  FileHandle=RECORD
47   link:MessagePtr;
48   port:MsgPortPtr;
49   type:MsgPortPtr;
50   buf:LONGINT;
51   pos:LONGINT;
52   end:LONGINT;
53   func1:LONGINT;
54   func2:LONGINT;
55   func3:LONGINT;
56   arg1:LONGINT;
57   arg2:LONGINT;
```

```
 1  END;
 2  DosPacket=RECORD
 3   link:MessagePtr;
 4   port:MsgPortPtr;
 5   type:LONGINT; (* Action *)
 6   res1:LONGINT; (* Status *)
 7   res2:LONGINT; (* Status2 *)
 8   arg1:LONGINT; (* BufAddr *)
 9   arg2:LONGINT;
10   arg3:LONGINT;
11   arg4:LONGINT;
12   arg5:LONGINT;
13   arg6:LONGINT;
14   arg7:LONGINT;
15  END;
16  DosPacketPtr=POINTER TO DosPacket;
17  StandardPacket=RECORD
18   msg:Message;
19   pkt:DosPacket;
20  END;
21
22 CONST
23   (* Mögliche Werte für DosPacket.type *)
24  nil=0;
25  getBlock=2;
26  setMap=4;
27  die=5;
28  event=6;
29  currentVolume=7;
30  locateObject=8;
31  renameDisk=9;
32  write=ORD('W');
33  read=ORD('R');
34  freeLock=15;
35  deleteObject=16;
36  renameObject=17;
37  moreCache=18;
38  copyDir=19;
39  waitChar=20;
40  setProtect=21;
41  createDir=22;
42  examineObject=23;
43  examineNext=24;
44  diskInfo=25;
45  info=26;
46  flush=27;
47  setComment=28;
48  parent=29;
49  timer=30;
50  inhibit=31;
51  diskType=32;
52  diskChange=33;
53  setDate=34;
54  screenMode=994;
55
56 TYPE
57  RootNodePtr=POINTER TO RootNode;
```

```
DosLibrary=RECORD
 lib:Library;
 root:RootNodePtr;
 gv:ADDRESS;
 a2:LONGINT;
 a5:LONGINT;
 a6:LONGINT;
END;
DosLibraryPtr=POINTER TO DosLibrary;
 (* dosLib:=ADR(Dos) *)
TaskArray=RECORD
 maxCli:LONGINT;
 cli:ARRAY [1..99] OF ProcessId;
END;
DeviceListPtr=BPOINTER TO DeviceList;
DosInfoPtr=BPOINTER TO DosInfo;
TaskArrayPtr=BPOINTER TO TaskArray;
RootNode=RECORD
 taskArray:TaskArrayPtr;
 consoleSegment:BPTR;
 time:Date;
 restartSeg:BPTR;
 info:DosInfoPtr;
 fileHandlerSegment:BPTR;
END;
DosInfo=RECORD
 mcName:BSTR;
 devInfo:DeviceListPtr;
 devices:BPTR;
 handlers:BPTR;
 netHand:ADDRESS;
END;
CommandLineInterface=RECORD
 result2:LONGINT;
 setName:BSTR;
 commandDir:BPTR; (* neuerdings command path ! *)
 returnCode:LONGINT;
 commandName:BSTR;
 failLevel:LONGINT;
 prompt:BSTR;
 standardInput:FileHandlePtr;
 currentInput:FileHandlePtr;
 commandFile:BSTR;
 interactive:LONGINT; (* LONGBOOLEAN *)
 background:LONGINT; (* LONGBOOLEAN *)
 currentOutput:FileHandlePtr;
 defaultStack:LONGINT;
 standardOutput:FileHandlePtr;
 module:BPTR;
END;
DeviceListType=(device,directory,volume);
DeviceList=RECORD
 next:DeviceListPtr;
 pad1,pad2,pad3:BYTE;
 type:DeviceListType;
 task:ProcessId;
 lock:FileLockPtr;
```

```
  CASE :DeviceListType OF
  | device,directory:
   handler:BSTR;
   stackSize:LONGINT;
   priority:LONGINT;
   startup:LONGINT;
   segList:BPTR;
   globVec:BPTR;
  | volume:
   volumeDate:Date;
   lockList:FileLockPtr;
   diskType:LONGINT;
   unused:LONGINT
  END;
  name:BSTR;
 END;
 FileLock=RECORD
  link:FileLockPtr;
  key:LONGINT;
  access:LONGINT;
  task:ProcessId;
  volume:DeviceListPtr;
 END;

PROCEDURE Close(file{1}:FileHandlePtr); CODE -36;
PROCEDURE CreateDir(name{1}:ADDRESS):FileLockPtr; CODE -120;
PROCEDURE CreateProc(
            name{1}:ADDRESS;
            pri{2}:LONGINT;
            segment{3}:BPTR;
            stackSize{4}:LONGINT):ProcessId; CODE -138;
PROCEDURE CurrentDir(
            lock{1}:FileLockPtr):FileLockPtr; CODE -126;
PROCEDURE DateStamp(v{1}:DatePtr); CODE -192;
PROCEDURE Delay(ticks{1}:LONGINT); CODE -198;
PROCEDURE DeleteFile(name{1}:ADDRESS):BOOLEAN; CODE -72;
PROCEDURE DeviceProc(name{1}:ADDRESS):ProcessId; CODE -174;
PROCEDURE DupLock(lock{1}:FileLockPtr):FileLockPtr; CODE -96;
PROCEDURE Examine(
            lock{1}:FileLockPtr;
            infoBlock{2}:FileInfoBlockPtr):BOOLEAN; CODE -102;
PROCEDURE Execute(
            commandString{1}:ADDRESS;
            input{2}:FileHandlePtr;
            output{3}:FileHandlePtr):LONGINT; CODE -222;
PROCEDURE Exit(returnCode{1}:LONGINT); CODE -144;
PROCEDURE ExNext(
            lock{1}:FileLockPtr;
            infoBlock{2}:FileInfoBlockPtr):BOOLEAN; CODE -108;
PROCEDURE IoErr():LONGINT; CODE -132;
(* PRIVATE *)
PROCEDURE GetPacket(wait{1}:LONGINT):DosPacketPtr; CODE -162;
PROCEDURE Info(
            lock{1}:FileLockPtr;
            parameterBlock{2}:InfoDataPtr):BOOLEAN; CODE -114;
PROCEDURE Input():FileHandlePtr; CODE -54;
PROCEDURE IsInteractive(
```

```
1             file{1}:FileHandlePtr):BOOLEAN; CODE -216;
2  PROCEDURE LoadSeg(name{1}:ADDRESS):BPTR; CODE -150;
3  PROCEDURE Lock(name{1}:ADDRESS;
4                  accessMode{2}:LONGINT):FileLockPtr; CODE -84;
5  PROCEDURE Seek(file{1}:FileHandlePtr;
6                  position{2}:LONGINT;
7                  mode{3}:LONGINT):LONGINT; CODE -66;
8  PROCEDURE SetComment(
9             name{1},comment{2}:ADDRESS):BOOLEAN; CODE -180;
10 PROCEDURE SetProtection(name{1}:ADDRESS;
11                         mask{2}:ProtectionFlagSet):BOOLEAN;
12 PROCEDURE Open(
13            name{1}:ADDRESS;
14            accessMode{2}:LONGINT):FileHandlePtr; CODE -30;
15 PROCEDURE Output():FileHandlePtr; CODE -60;
16 PROCEDURE ParentDir(
17            lock{1}:FileLockPtr):FileLockPtr; CODE -210;
18 (*PRIVATE*)
19 PROCEDURE QueuePacket(
20            packet{1}:DosPacketPtr):LONGINT; CODE -168;
21 PROCEDURE Read(file{1}:FileHandlePtr;
22                buffer{2}:ADDRESS;
23                length{3}:LONGINT):LONGINT; CODE -42;
24 PROCEDURE Rename(
25            oldName{1},newName{2}:ADDRESS):BOOLEAN; CODE -78;
26 PROCEDURE UnLoadSeg(segment{1}:BPTR); CODE -156;
27 PROCEDURE UnLock(lock{1}:FileLockPtr); CODE -90;
28 PROCEDURE WaitForChar(file{1}:FileHandlePtr;
29                       timeout{2}:LONGINT):BOOLEAN; CODE -204;
30 PROCEDURE Write(file{1}:FileHandlePtr;
31                 buffer{2}:ADDRESS;
32                 length{3}:LONGINT):LONGINT; CODE -48;
33 END Dos.
34
```

Exec

```
 5 DEFINITION MODULE Exec {"exec.library",33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS,BITSET,BYTE,LONGSET,WORD;
 9
10 IMPORT Hardware;
11
12 TYPE
13  Byte=[-128..127];
14  UByte=[0..255];
15  NodeType=(
16   unknown,task,interrupt,device,msgPort,message,freeMsg,
17   replyMsg,resource,library,memory,softInt,font,process,
18   semaphore,signalSem
19  );
20  NodePtr=POINTER TO Node;
21  Node=RECORD
22   succ:NodePtr;
23   pred:NodePtr;
24   type:NodeType;
25   pri:Byte;
26   name:ADDRESS;
27  END;
28  MinNodePtr=POINTER TO MinNode;
29  MinNode=RECORD
30   succ:MinNodePtr;
31   pred:MinNodePtr;
32  END;
33  List=RECORD
34   head:NodePtr;
35   tail:NodePtr;
36   tailPred:NodePtr;
37   type:NodeType;
38   pad:BYTE;
39  END;
40  ListPtr=POINTER TO List;
41  MinList=RECORD
42   head:MinNodePtr;
43   tail:MinNodePtr;
44   tailPred:MinNodePtr;
45  END;
46  MinListPtr=POINTER TO MinList;
47  Interrupt=RECORD
48   node:Node;
49   data:ADDRESS;
50   code:PROC;
51  END;
52  InterruptPtr=POINTER TO Interrupt;
53  IntVector=RECORD
54   data:ADDRESS;
55   code:PROC;
56   node:NodePtr;
57  END;
```

```
SoftIntList=RECORD
 list:List;
 pad:WORD;
END;
MemReqs=(
 public,chip,fast,mr3,mr4,mr5,mr6,mr7,
 mr8,mr9,mr10,mr11,mr12,mr13,mr14,mr15,memClear,largest
);
MemReqSet=SET OF MemReqs;
MemTypeSet=SET OF [public..mr15];
MemChunkPtr=POINTER TO MemChunk;
MemChunk=RECORD
 next:MemChunkPtr;
 bytes:LONGCARD;
END;
MemHeader=RECORD
 node:Node;
 attributes:MemTypeSet;
 first:MemChunkPtr;
 lower:ADDRESS;
 upper:ADDRESS;
 free:LONGCARD;
END;
MemHeaderPtr=POINTER TO MemHeader;
MemEntry=RECORD
 CASE :INTEGER OF
 | 1:reqs:MemReqSet
 | 2:addr:ADDRESS
 END;
 length:LONGCARD;
END;
MemList=RECORD
 node:Node;
 numEntries:CARDINAL;
(*me:ARRAY [0..munEntries-1] OF MemEntry;*)
END;
MemListPtr=POINTER TO MemList;
MsgPortAction=(signal, softint, ignore);
TaskPtr=POINTER TO Task;
MsgPort=RECORD
 node:Node;
 CASE flags:MsgPortAction OF
 | signal:
  sigBit:UByte;
  sigTask:TaskPtr;
 | softint:
  pad0:BYTE;
  softInt:InterruptPtr;
 | ignore:
  pad1:BYTE;
  pad2:ADDRESS
 END;
 msgList:List;
END;
MsgPortPtr=POINTER TO MsgPort;
Message=RECORD
 node:Node;
```

```
 1    replyPort:MsgPortPtr;
 2    length:CARDINAL;
 3   END;
 4   MessagePtr=POINTER TO Message;
 5   TaskFlags=(
 6    procTime,tf1,tf2,tf3,stackChk,exception,switch,launch
 7   );
 8   TaskFlagSet=SET OF TaskFlags;
 9   TaskState=(inval,added,run,ready,wait,except,removed);
10   Task=RECORD
11    node:Node;
12    flags:TaskFlagSet;
13    state:TaskState;
14    idNestCnt:BYTE;
15    tdNestCnt:BYTE;
16    sigAlloc:LONGSET;
17    sigWait:LONGSET;
18    sigRecvd:LONGSET;
19    sigExcept:LONGSET;
20    trapAlloc:BITSET;
21    trapAble:BITSET;
22    exceptData:ADDRESS;
23    exceptCode:PROC;
24    trapData:ADDRESS;
25    trapCode:PROC;
26    spReg:ADDRESS;
27    spLower:ADDRESS;
28    spUpper:ADDRESS;
29    switch:PROC;
30    launch:PROC;
31    memEntry:List;
32    userData:ADDRESS;
33   END;
34
35  CONST
36     (* Vordefinierte Signalnummern *)
37    sigAbort=0;
38    sigChild=1;
39    sigBlit=4;
40    sigDos=8;
41
42    vectSize=6;
43    reserved=4;
44    base=-vectSize;
45    userDef=base-reserved*vectSize-30;
46    nonStd=userDef;
47    extFunc=-24;
48    expunge=-18;
49    close=-12;
50    open=-6;
51
52  TYPE
53    LibFlags=(summing,changed,sumUsed,delExp);
54    LibFlagSet=SET OF LibFlags;
55    Library=RECORD
56     node:Node;
57     flags:LibFlagSet;
```

```
  pad:BYTE;
  negSize:CARDINAL;
  posSize:CARDINAL;
  version:CARDINAL;
  revision:CARDINAL;
  idString:ADDRESS;
  sum:LONGCARD;
  openCnt:CARDINAL;
 END;
 LibraryPtr=POINTER TO Library;
 Device=RECORD
  library:Library;
 END;
 DevicePtr=POINTER TO Device;
 UnitFlags=(active,inTask);
 UnitFlagSet=SET OF UnitFlags;
 UnitPtr=POINTER TO Unit;
 Unit=RECORD
  msgPort:MsgPortPtr;
  flags:UnitFlagSet;
  pad:BYTE;
  openCnt:CARDINAL;
 END;

CONST
   (* Standardbefehle für IORequest.command *)
 invalid=0;
 reset=1;
 read=2;
 write=3;
 update=4;
 clear=5;
 stop=6;
 start=7;
 flush=8;
 nonstd=9;
   (* Offstes der Devicespezifischen Funktionen  *)
 abortIO=-36;
 beginIO=-30;

TYPE
 IOFlagSet=SET OF [0..7];

CONST
 quick=IOFlagSet{0};

TYPE
 IORequest=RECORD
  message:Message;
  device:DevicePtr;
  unit:UnitPtr;
  command:CARDINAL;
  flags:IOFlagSet;
  error:Byte;
 END;
 IORequestPtr=POINTER TO IORequest;
 IOStdReq=RECORD
```

```
 1   message:Message;
 2   device:DevicePtr;
 3   unit:UnitPtr;
 4   command:CARDINAL;
 5   flags:IOFlagSet;
 6   error:Byte;
 7   actual:LONGCARD;
 8   length:LONGCARD;
 9   data:ADDRESS;
10   offset:LONGCARD;
11  END;
12  IOStdReqPtr=POINTER TO IOStdReq;
13  Semaphore=RECORD
14   msgPort:MsgPort;
15   bids:INTEGER;
16  END;
17  SemaphoreRequest=RECORD
18   link:MinNode;
19   waiter:TaskPtr;
20  END;
21  SignalSemaphore=RECORD
22   link:Node;
23   nestCount:INTEGER;
24   waitQueue:MinList;
25   multipleLink:SemaphoreRequest;
26   owner:TaskPtr;
27   queueCount:INTEGER;
28  END;
29  SignalSemaphorePtr=POINTER TO SignalSemaphore;
30  ResidentFlags=(coldstart,rf1,rf2,rf3,rf4,rf5,rf6,autoinit);
31  ResidentFlagSet=SET OF ResidentFlags;
32  ResidentPtr=POINTER TO Resident;
33  Resident=RECORD
34   matchWord:CARDINAL;
35   matchTag:ResidentPtr;
36   endSkip:ADDRESS;
37   flags:ResidentFlagSet;
38   version:UByte;
39   type:NodeType;
40   pri:Byte;
41   name:ADDRESS;
42   idString:ADDRESS;
43   init:ADDRESS;
44  END;
45
46 CONST
47  matchword=04AFCH;
48
49 TYPE
50  AttnFlags=(
51   m68010,m68020,af2,af3,m68881,af5,af6,af7,
52   reserved8,reserved9
53  );
54  AttnFlagSet=SET OF AttnFlags;
55  ExecBase=RECORD
56   libNode:Library;
57   softVer:CARDINAL;
```

```
  lowMemChkSum:INTEGER;
  chkBase:LONGCARD;
  coldCapture:ADDRESS;
  coolCapture:ADDRESS;
  warmCapture:ADDRESS;
  sysStkUpper:ADDRESS;
  sysStkLower:ADDRESS;
  maxLocMem:LONGCARD;
  debugEntry:ADDRESS;
  debugData:ADDRESS;
  alertData:ADDRESS;
  maxExtMem:ADDRESS;
  chkSum:CARDINAL;
  intVects:ARRAY Hardware.IntFlags OF IntVector;
  thisTask:TaskPtr;
  idleCount:LONGCARD;
  dispCount:LONGCARD;
  quantum:CARDINAL;
  elapsed:CARDINAL;
  sysFlags:CARDINAL;
  idNestCnt:BYTE;
  tdNestCnt:BYTE;
  attnFlags:AttnFlagSet;
  attnResched:CARDINAL;
  resModules:ADDRESS;
  taskTrapCode:PROC;
  taskExceptCode:PROC;
  taskExitCode:PROC;
  taskSigAlloc:LONGSET;
  taskTrapAlloc:BITSET;
  memList:List;
  resourceList:List;
  deviceList:List;
  intrList:List;
  libList:List;
  portList:List;
  taskReady:List;
  taskWait:List;
  softInts:ARRAY [0..4] OF SoftIntList;
  lastAlert:ARRAY [0..3] OF LONGINT;
  vBlankFrequency:UByte;
  powerSupplyFrequency:UByte;
  semaphoreList:List;
  kickMemPtr:ADDRESS;
  kickTagPtr:ADDRESS;
  kickCheckSum:ADDRESS;
  execBaseReserved:ARRAY [0..9] OF BYTE;
  execBaseNewReserved:ARRAY [0..19] OF BYTE;
END;
ExecBasePtr=POINTER TO ExecBase;

VAR
 execBase[4]:ExecBasePtr;

PROCEDURE AbortIO(iORequest{9}:ADDRESS); CODE -480;
PROCEDURE AddDevice(device{9}:DevicePtr); CODE -432;
PROCEDURE AddHead(list{8}:ListPtr;
```

```
1                    node{9}:ADDRESS); CODE -240;
2 PROCEDURE AddIntServer(intNum{0}:Hardware.IntFlags;
3                        interrupt{9}:InterruptPtr); CODE -168;
4 PROCEDURE AddLibrary(library{9}:LibraryPtr); CODE -396;
5 PROCEDURE AddMemList(size{0}:LONGINT;
6                       attributes{1}:MemReqSet;
7                       pri{2}:LONGINT;
8                       base{8}:ADDRESS;
9                       name{9}:ADDRESS):LONGINT; CODE -618;
10 PROCEDURE AddPort(port{9}:MsgPortPtr); CODE -354;
11 PROCEDURE AddResource(resource{9}:ADDRESS); CODE -486;
12 PROCEDURE AddSemaphore(
13              signalSemaphore{8}:SignalSemaphorePtr); CODE -600;
14 PROCEDURE AddTail(list{8}:ListPtr;
15                   node{9}:ADDRESS); CODE -246;
16 PROCEDURE AddTask(task{9}:TaskPtr;
17                   initialPC{10}:ADDRESS;
18                   finalPC{11}:ADDRESS); CODE -282;
19 PROCEDURE Alert(alertNum{7}:LONGINT;
20                 parameters{13}:ADDRESS); CODE -108;
21 PROCEDURE AllocAbs(byteSize{0}:LONGINT;
22                    location{9}:ADDRESS):ADDRESS; CODE -204;
23 PROCEDURE Allocate(freeList{8}:MemHeaderPtr;
24                    byteSize{0}:LONGINT):ADDRESS; CODE -186;
25 PROCEDURE AllocEntry(
26              memList1{8}:MemListPtr):MemListPtr; CODE -222;
27 PROCEDURE AllocMem(
28              byteSize{0}:LONGINT;
29              requirements{1}:MemReqSet):ADDRESS; CODE -198;
30 PROCEDURE AllocSignal(
31              signalNum1{0}:LONGINT):LONGINT; CODE -330;
32 PROCEDURE AllocTrap(trapNum1{0}:LONGINT):LONGINT; CODE -342;
33 PROCEDURE AttemptSemaphore(
34              signalSemaphore{8}:SignalSemaphorePtr
35             ):LONGINT; CODE -576;
36 PROCEDURE AvailMem(
37              requirements{1}:MemReqSet):LONGINT; CODE -216;
38 PROCEDURE Cause(interrupt{9}:InterruptPtr); CODE -180;
39 PROCEDURE CheckIO(ioRequest{9}:ADDRESS):ADDRESS; CODE -468;
40 PROCEDURE CloseDevice(ioRequest{9}:ADDRESS); CODE -450;
41 PROCEDURE CloseLibrary(library{9}:LibraryPtr); CODE -414;
42 PROCEDURE CopyMem(source{8}:ADDRESS;
43                   dest{9}:ADDRESS;
44                   size{0}:LONGINT); CODE -624;
45 PROCEDURE CopyMemQuick(source{8}:ADDRESS;
46                         dest{9}:ADDRESS;
47                         size{0}:LONGINT); CODE -630;
48 PROCEDURE Deallocate(freeList{8}:MemHeaderPtr;
49                       memoryBlock{9}:ADDRESS;
50                       byteSize{0}:LONGINT); CODE -192;
51 PROCEDURE Debug(); CODE -114;
52 PROCEDURE Disable(); CODE -120;
53 (* PRIVATE *) PROCEDURE Dispatch(); CODE -60;
54 PROCEDURE DoIO(ioRequest{9}:ADDRESS); CODE -456;
55 PROCEDURE Enable(); CODE -126;
56 PROCEDURE Enqueue(list{8}:ListPtr;
57                   node{9}:ADDRESS); CODE -270;
```

```
1  (* PRIVATE *) PROCEDURE Exception(); CODE -66;
2  (* PRIVATE *) PROCEDURE ExitIntr(); CODE -36;
3  PROCEDURE FindName(start{8}:ADDRESS;
4                     name{9}:ADDRESS):ADDRESS; CODE -276;
5  PROCEDURE FindPort(name{9}:ADDRESS):MsgPortPtr; CODE -390;
6  PROCEDURE FindResident(
7             name{9}:ADDRESS):ResidentPtr; CODE -96;
8  PROCEDURE FindSemaphore(
9             name{8}:ADDRESS):SignalSemaphorePtr; CODE -594;
10 PROCEDURE FindTask(name{9}:ADDRESS):TaskPtr; CODE -294;
11 PROCEDURE Forbid(); CODE -132;
12 PROCEDURE FreeEntry(memList{8}:MemListPtr); CODE -228;
13 PROCEDURE FreeMem(memoryBlock{9}:ADDRESS;
14                   byteSize{0}:LONGINT); CODE -210;
15 PROCEDURE FreeSignal(signalNum{0}:LONGINT); CODE -336;
16 PROCEDURE FreeTrap(trapNum{0}:LONGINT); CODE -348;
17 PROCEDURE GetCC():BITSET; CODE -528;
18 PROCEDURE GetMsg(port{8}:MsgPortPtr):ADDRESS; CODE -372;
19 PROCEDURE InitCode(startClass{0}:ResidentFlagSet;
20                    version{1}:LONGINT); CODE -72;
21 PROCEDURE InitResident(resident{9}:ResidentPtr;
22                        segList{1}:ADDRESS); CODE -102;
23 PROCEDURE InitSemaphore(
24            signalSemaphore{8}:SignalSemaphorePtr); CODE -558;
25 PROCEDURE InitStruct(initTable{9}:ADDRESS;
26                      memory{10}:ADDRESS;
27                      size{0}:CARDINAL); CODE -78;
28 PROCEDURE Insert(list{8}:ListPtr;
29                 node{9}:ADDRESS;
30                 listNode{10}:ADDRESS); CODE -234;
31 PROCEDURE MakeFunctions(target{8}:ADDRESS;
32                         functArray{9}:ADDRESS;
33                         functDispBase{10}:ADDRESS); CODE -90;
34 PROCEDURE MakeLibrary(
35            vectors{8}:ADDRESS;
36            structure{9}:ADDRESS;
37            init{10}:ADDRESS;
38            dataSize{0}:LONGINT;
39            segList{1}:ADDRESS):LibraryPtr; CODE -84;
40 PROCEDURE ObtainSemaphore(
41            signalSemaphore{8}:SignalSemaphorePtr); CODE -564;
42 PROCEDURE ObtainSemaphoreList(list{8}:ListPtr); CODE -582;
43 PROCEDURE OldOpenLibrary(
44            libName{9}:ADDRESS):LibraryPtr; CODE -408;
45 PROCEDURE OpenDevice(devName{8}:ADDRESS;
46                      unitNumber{0}:LONGINT;
47                      ioRequest{9}:ADDRESS;
48                      flags{1}:LONGSET); CODE -444;
49 PROCEDURE OpenLibrary(
50            libName{9}:ADDRESS;
51            version{0}:LONGINT):LibraryPtr; CODE -552;
52 PROCEDURE OpenResource(resName{9}:ADDRESS):ADDRESS; CODE -498;
53 PROCEDURE Permit(); CODE -138;
54 PROCEDURE Procure(
55            semaphore{8}:MsgPortPtr;
56            bidMessage{9}:MsgPortPtr):LONGINT; CODE -540;
57 PROCEDURE PutMsg(port{8}:MsgPortPtr;
```

```
                    message{9}:ADDRESS); CODE -366;
(* PRIVATE *) PROCEDURE RawDoFmt(
                             formatString{8}:ADDRESS;
                             dataStream{9}:ADDRESS;
                             putChProc{10}:ADDRESS;
                             putChData{11}:ADDRESS); CODE -522;
(* PRIVATE *) PROCEDURE RawIOInit(); CODE -504;
(* PRIVATE *) PROCEDURE RawMayGetChar():CHAR; CODE -510;
(* PRIVATE *) PROCEDURE RawPutChar(ch{0}:CHAR); CODE -516;
PROCEDURE ReleaseSemaphore(
             signalSemaphore{8}:SignalSemaphorePtr); CODE -570;
PROCEDURE ReleaseSemaphoreList(list{8}:ListPtr); CODE -588;
PROCEDURE RemDevice(device{9}:DevicePtr):LONGINT; CODE -438;
PROCEDURE RemHead(list{8}:ListPtr):ADDRESS; CODE -258;
PROCEDURE RemIntServer(intNum{0}:Hardware.IntFlags;
                       interrupt{9}:InterruptPtr); CODE -174;
PROCEDURE RemLibrary(
             library{9}:LibraryPtr):LONGINT; CODE -402;
PROCEDURE Remove(node{9}:ADDRESS); CODE -252;
PROCEDURE RemPort(port{9}:MsgPortPtr); CODE -360;
PROCEDURE RemResource(resource{9}:ADDRESS); CODE -492;
PROCEDURE RemSemaphore(
             signalSemaphore{8}:SignalSemaphorePtr); CODE -606;
PROCEDURE RemTail(list{8}:ListPtr):ADDRESS; CODE -264;
PROCEDURE RemTask(task{9}:TaskPtr); CODE -288;
PROCEDURE ReplyMsg(message{9}:ADDRESS); CODE -378;
(* PRIVATE *) PROCEDURE Reschedule(); CODE -48;
(* PRIVATE *) PROCEDURE Schedule(); CODE -42;
PROCEDURE SendIO(ioRequest{9}:ADDRESS); CODE -462;
PROCEDURE SetExcept(newSignals{0}:LONGSET;
                    signalMask{1}:LONGSET):LONGSET; CODE -312;
PROCEDURE SetFunction(
             library{9}:LibraryPtr;
             funcOffset{8}:INTEGER;
             funcEntry{0}:ADDRESS):ADDRESS; CODE -420;
PROCEDURE SetIntVector(
             intNumber{0}:Hardware.IntFlags;
             interrupt{9}:InterruptPtr):InterruptPtr; CODE -162;
PROCEDURE SetSignal(newSignals{0}:LONGSET;
                    signalMask{1}:LONGSET):LONGSET; CODE -306;
PROCEDURE SetSR(newSR{0}:BITSET;
                mask{1}:BITSET):BITSET; CODE -144;
PROCEDURE SetTaskPri(task{9}:TaskPtr;
                     priority{0}:Byte):Byte; CODE -300;
PROCEDURE Signal(task{9}:TaskPtr;
                 signals{0}:LONGSET); CODE -324;
PROCEDURE SumKickData(); CODE -612;
PROCEDURE SumLibrary(library{9}:LibraryPtr); CODE -426;
PROCEDURE SuperState():ADDRESS; CODE -150;
(* PRIVATE *) PROCEDURE Supervisor(); CODE -30;
(* PRIVATE *) PROCEDURE Switch(); CODE -54;
PROCEDURE TypeOfMem(address{9}:ADDRESS):MemReqSet; CODE -534;
PROCEDURE UserState(sysStack{0}:ADDRESS); CODE -156;
PROCEDURE Vacate(semaphore{8}:MsgPortPtr); CODE -546;
PROCEDURE Wait(signalSet{0}:LONGSET):LONGSET; CODE -318;
PROCEDURE WaitIO(ioRequest{9}:ADDRESS); CODE -474;
PROCEDURE WaitPort(port{8}:MsgPortPtr); CODE -384;
```

```
1
2 END Exec.
3
```

ExecSupport

```
 5 (* $N- *)
 6 DEFINITION MODULE ExecSupport;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Exec IMPORT
11  Byte,ListPtr,IOStdReqPtr,MsgPortPtr,TaskPtr;
12
13 PROCEDURE NewList(list{8}:ListPtr);
14 PROCEDURE BeginIO(ioRequest{9}:ADDRESS);
15 PROCEDURE AbortIO(ioRequest{9}:ADDRESS);
16 PROCEDURE CreatePort(portName:ADDRESS;
17                       priority:Byte):MsgPortPtr;
18 PROCEDURE DeletePort(msgPort:MsgPortPtr);
19 PROCEDURE CreateExtIO(ioReplyPort:MsgPortPtr;
20                        size:INTEGER):ADDRESS;
21 PROCEDURE DeleteExtIO(extIOReq:ADDRESS);
22 PROCEDURE CreateStdIO(ioReplyPort:MsgPortPtr):IOStdReqPtr;
23 PROCEDURE DeleteStdIO(ioStdReq:IOStdReqPtr);
24 PROCEDURE CreateTask(taskName:ADDRESS;
25                       priority:Byte;
26                       initPC:ADDRESS;
27                       stackSize:LONGINT):TaskPtr;
28 PROCEDURE DeleteTask(t:TaskPtr);
29
30 END ExecSupport.
31
```

Expansion

```
 5 DEFINITION MODULE Expansion {"expansion.library",33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS,BYTE,LONGSET;
 9 FROM Exec IMPORT
10  Node,UByte;
11
12 TYPE
13  ExpansionRom=RECORD
14   type:UByte;
15   product:UByte;
16   flags:UByte;
17   reserved03:UByte;
18   manufacturer:CARDINAL;
19   serialNumber:LONGCARD;
20   initDiagVec:CARDINAL;
21   reserved0c:UByte;
22   reserved0d:UByte;
23   reserved0e:UByte;
24   reserved0f:UByte
25  END;
26  ExpansionControl=RECORD
27   interrupt:UByte;
28   reserved11:UByte;
29   baseAddress:UByte;
30   shutup:UByte;
31   reserved14:UByte;
32   reserved15:UByte;
33   reserved16:UByte;
34   reserved17:UByte;
35   reserved18:UByte;
36   reserved19:UByte;
37   reserved1a:UByte;
38   reserved1b:UByte;
39   reserved1c:UByte;
40   reserved1d:UByte;
41   reserved1e:UByte;
42   reserved1f:UByte;
43  END;
44
45 CONST
46  slotSize=10000H;
47  slotMask=0FFFFH;
48  slotShift=16;
49  expansionBase=0E80000H;
50  expansionSize=080000H;
51  expansionSlots=8;
52  memoryBase=200000H;
53  memorySize=800000H;
54  memorySlots=128;
55  typrMask=0C0H;
56  typeBit=6;
57  typeSize=2;
```

```
 1  newBoard=0C0H;
 2  memMask=07H;
 3  memBit=0;
 4  memSize=3;
 5  chainedConfig=3;
 6  diagValid=4;
 7  memList=5;
 8  memSpace=7;
 9  noShutup=6;
10
11  intena=1;
12  reset=3;
13  int2pend=4;
14  int6pend=5;
15  int7pend=6;
16  interrupting=7;
17
18 TYPE
19  DiagArea=RECORD
20   config:UByte;
21   flags:UByte;
22   size:CARDINAL;
23   diagPoint:CARDINAL;
24   bootPoint:CARDINAL;
25   name:CARDINAL;
26   reserved01:CARDINAL;
27   reserved02:CARDINAL
28  END;
29
30 CONST
31  busWidth=0C0H;
32  nibbleWide=0;
33  byteWide=040H;
34  wordWide=080H;
35  bootTime=030H;
36  never=0;
37  configTime=010H;
38  bindTime=020H;
39
40 TYPE
41  ConfigDevPtr=POINTER TO ConfigDev;
42  ConfigDev=RECORD
43   node:Node;
44   flags:UByte;
45   pad:UByte;
46   rom:ExpansionRom;
47   boardAddr:ADDRESS;
48   boardSize:ADDRESS;
49   slotAddr:CARDINAL;
50   slotSize:CARDINAL;
51   driver:ADDRESS;
52   nextCD:ConfigDevPtr;
53   unused:ARRAY [0..3] OF LONGINT;
54  END;
55
56 CONST
57  shutup=0;
```

```
 1  configMe=1;
 2
 3 TYPE
 4  CurrentBinding=RECORD
 5   configDev:ConfigDevPtr;
 6   fileName:ADDRESS;
 7   productString:ADDRESS;
 8   toolTypes:ADDRESS;
 9  END;
10  CurrentBindingPtr=POINTER TO CurrentBinding;
11
12 PROCEDURE AddConfigDev(configDev{8}:ConfigDevPtr); CODE -30;
13 PROCEDURE AddDosNode(
14             bootPri{0}:LONGINT;
15             flags{1}:LONGSET;
16             deviceNode{8}:ADDRESS):LONGINT; CODE -150;
17 PROCEDURE AllocBoardMem(
18             slotSpec{0}:LONGINT):LONGINT; CODE -42;
19 PROCEDURE AllocConfigDev():ConfigDevPtr; CODE -48;
20 PROCEDURE AllocExpansionMem(
21             numSlots{0}:LONGINT;
22             slotAlign{1}:LONGINT;
23             slotOffset{2}:LONGINT):LONGINT; CODE -54;
24 PROCEDURE ConfigBoard(
25             board{8}:ADDRESS;
26             configDev{9}:ConfigDevPtr):LONGINT; CODE -60;
27 PROCEDURE ConfigChain(baseAddr{8}:ADDRESS); CODE -66;
28 PROCEDURE expansionUnused(); CODE -36;
29 PROCEDURE FindConfigDev(oldConfigDev{8}:ConfigDevPtr;
30                         manufacturer{0}:LONGINT;
31                         product{1}:LONGINT); CODE -72;
32 PROCEDURE FreeBoardMem(startSlot{0}:LONGINT;
33                       slotSpec{1}:LONGINT); CODE -78;
34 PROCEDURE FreeConfigDev(configDev{8}:ConfigDevPtr); CODE -84;
35 PROCEDURE FreeExpansionMem(startSlot{0}:LONGINT;
36                            numSlots{1}:LONGINT); CODE -90;
37 PROCEDURE GetCurrentBinding(
38             currentBinding{8}:CurrentBindingPtr;
39             size{0}:LONGINT):LONGINT; CODE -138;
40 PROCEDURE MakeDosNode(
41             parameterPkt{0}:ADDRESS):ADDRESS; CODE -144;
42 PROCEDURE ObtainConfigBinding(); CODE -120;
43 PROCEDURE ReadExpansionByte(board{8}:ADDRESS;
44                             offset{0}:LONGINT):BYTE; CODE -96;
45 PROCEDURE ReadExpansionRom(
46             board{8}:ADDRESS;
47             configDev{9}:ConfigDevPtr):LONGINT; CODE -102;
48 PROCEDURE ReleaseConfigBinding(); CODE -126;
49 PROCEDURE RemConfigDev(configDev{8}:ConfigDevPtr); CODE -108;
50 PROCEDURE SetCurrentBinding(
51             currentBinding{8}:CurrentBindingPtr;
52             size{0}:LONGINT); CODE -132;
53 PROCEDURE WriteExpansionByte(board{8}:ADDRESS;
54                              offset{0}:LONGINT;
55                              byte{1}:BYTE); CODE -114;
56
57 END Expansion.
```

1

GamePort

```
(* $M- *)
DEFINITION MODULE GamePort;

FROM Exec IMPORT
 nonstd;

CONST
 gamePortName="gameport.device";
 readEvent=nonstd+0;
 askCType=nonstd+1;
 setCType=nonstd+2;
 askTrigger=nonstd+3;
 setTrigger=nonstd+4;
 allocated=-1;
 errSetCType=1;

TYPE
 Keys=(downKeys,upKeys,k2,k3,k4,k5,k6,k7,k8);
 KeySet=SET OF Keys;
 GamePortTrigger=RECORD
  keys:KeySet;
  timeout:CARDINAL;
  xDelta:CARDINAL;
  yDelta:CARDINAL;
 END;
 Controller=(noController,mouse,relJoystick,absJoystick);

END GamePort.
```

GfxMacros

```
 5 (* $N- *)
 6 DEFINITION MODULE GfxMacros;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS, BYTE;
10 FROM Graphics IMPORT
11  BobPtr, RastPortPtr, UCopListPtr;
12
13 PROCEDURE InitAnimate(animKey:ADDRESS);
14 PROCEDURE RemBob(bob:BobPtr);
15 PROCEDURE RasSize(w, h:CARDINAL):CARDINAL;
16 PROCEDURE OnDisplay;
17 PROCEDURE OffDisplay;
18 PROCEDURE OnSprite;
19 PROCEDURE OffSprite;
20 PROCEDURE OnVBlank;
21 PROCEDURE OffVBlank;
22 PROCEDURE SetOPen(rp:RastPortPtr; pen:CARDINAL);
23 PROCEDURE SetDrPt(rp:RastPortPtr; pattern:CARDINAL);
24 PROCEDURE SetWrMsk(rp:RastPortPtr; mask:BYTE);
25 PROCEDURE SetAfPen(rp:RastPortPtr;
26                    areaPattern:ADDRESS; size:CARDINAL);
27 PROCEDURE BndryOff(rp:RastPortPtr);
28 PROCEDURE CINIT(c:UCopListPtr; n:LONGINT);
29 PROCEDURE CMOVE(c:UCopListPtr; a:ADDRESS; b:INTEGER);
30 PROCEDURE CWAIT(c:UCopListPtr; a, b:INTEGER);
31 PROCEDURE CEND(c:UCopListPtr; a, b:INTEGER);
32 PROCEDURE DrawCircle(rp:RastPortPtr; cx, cy, r:INTEGER);
33 PROCEDURE AreaCircle(rp:RastPortPtr; cx, cy, r:INTEGER);
34
35 END GfxMacros.
36
```

Graphics

```
 5 DEFINITION MODULE Graphics {"graphics.library", 33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS,BITSET,BYTE,LONGSET,WORD;
 9 FROM Hardware IMPORT
10  bltnodePtr;
11 FROM Exec IMPORT
12  Interrupt,Library,List,Message,MinList,Node,SignalSemaphore,
13  SignalSemaphorePtr,TaskPtr,UByte;
14
15 TYPE
16  AnimCompPtr=POINTER TO AnimComp;
17  AnimObPtr=POINTER TO AnimOb;
18  AreaInfoPtr=POINTER TO AreaInfo;
19  BitMapPtr=POINTER TO BitMap;
20  BobPtr=POINTER TO Bob;
21  ClipRectPtr=POINTER TO ClipRect;
22  CollTablePtr=POINTER TO CollTable;
23  ColorMapPtr=POINTER TO ColorMap;
24  CopinitPtr=POINTER TO Copinit;
25  CopInsPtr=POINTER TO CopIns;
26  CopListPtr=POINTER TO CopList;
27  CprlistPtr=POINTER TO Cprlist;
28  DBufPacketPtr=POINTER TO DBufPacket;
29  GelsInfoPtr=POINTER TO GelsInfo;
30  IsrvstrPtr=POINTER TO IsrvstrPtr;
31  LayerPtr=POINTER TO Layer;
32  LayerInfoPtr=POINTER TO LayerInfo;
33  RasInfoPtr=POINTER TO RasInfo;
34  RastPortPtr=POINTER TO RastPort;
35  RectanglePtr=POINTER TO Rectangle;
36  RegionPtr=POINTER TO Region;
37  RegionRectanglePtr=POINTER TO RegionRectangle;
38  SimpleSpritePtr=POINTER TO SimpleSprite;
39  TextAttrPtr=POINTER TO TextAttr;
40  TextFontPtr=POINTER TO TextFont;
41  TmpRasPtr=POINTER TO TmpRas;
42  UCopListPtr=POINTER TO UCopList;
43  ViewPtr=POINTER TO View;
44  ViewPortPtr=POINTER TO ViewPort;
45  VSpritePtr=POINTER TO VSprite;
46
47  Rectangle=RECORD
48   minX:INTEGER;
49   minY:INTEGER;
50   maxX:INTEGER;
51   maxY:INTEGER;
52  END;
53  Layer=RECORD
54   front:LayerPtr;
55   back:LayerPtr;
56   clipRect:ClipRectPtr;
57   rp:RastPortPtr;
```

```
  bounds:Rectangle;
  reserved:ARRAY [0..3] OF BYTE;
  priority:CARDINAL;
  flags:CARDINAL;
  superBitMap:BitMapPtr;
  superClipRect:ClipRectPtr;
  window:ADDRESS;
  scrollX:INTEGER;
  scrollY:INTEGER;
  cr:ClipRectPtr;
  cr2:ClipRectPtr;
  crnew:ClipRectPtr;
  superSaveClipRects:ClipRectPtr;
  cliprects:ClipRectPtr;
  layerInfo:LayerInfoPtr;
  lock:SignalSemaphore;
  reserved3:ARRAY [0..7] OF BYTE;
  clipRegion:RegionPtr;
  saveClipRects:RegionPtr;
  reserved2:ARRAY [0..21] OF BYTE;
  damageList:RegionPtr;
 END;
 ClipRect=RECORD
  next:ClipRectPtr;
  prev:ClipRectPtr;
  lobs:LayerPtr;
  bitMap:BitMapPtr;
  bounds:Rectangle;
  p1:ClipRectPtr;
  p2:ClipRectPtr;
  reserved:LONGINT;
  flags:LONGINT;
 END;

CONST
 needsNoConcealedRasters=01H;
 isLessX=1;
 isLessY=2;
 isGrtrX=4;
 isGrtrY=8;
 borderHit=0;
 topHit=1;
 bottomHit=2;
 leftHit=4;
 rightHit=8;

CONST
 move=0;
 wait=1;
 next=2;
 sht=14;
 lof=15;

TYPE
 CopIns=RECORD
  CASE opCode:CARDINAL OF
  | move:
```

```
    destAddr:INTEGER;
    destData:INTEGER;
   | wait:
    vWaitPos:INTEGER;
    hWaitPos:INTEGER;
   | next:
    nxtlist:CopListPtr;
  END;
 END;
 Cprlist=RECORD
  next:CprlistPtr;
  start:ADDRESS;
  maxCount:INTEGER;
 END;
 CopList=RECORD
  next:CopListPtr;
  copList:CopListPtr;
  viewPort:ViewPortPtr;
  copIns:CopInsPtr;
  copPtr:CopInsPtr;
  copLStart:ADDRESS;
  copSStart:ADDRESS;
  count:INTEGER;
  maxCount:INTEGER;
  dyOffset:INTEGER;
 END;
 UCopList=RECORD
  next:UCopListPtr;
  firstCopList:CopListPtr;
  copList:CopListPtr;
 END;
 Copinit=RECORD
  diagstrt:ARRAY [0..3] OF CARDINAL;
  sprstrtup:ARRAY [0..(2*8*2)+2+(2*2)+2-1] OF CARDINAL;
  sprstop:ARRAY [0..1] OF CARDINAL;
 END;

CONST
 interlace=04H;
 pf2pri=40H;
 colorOn=200H;
 dblpf=400H;
 holdnmodify=800H;
 m640=08000H;
 plnCntMsk=07H;
 plnCntShft=12;

 fineScroll=0FH;
 fineScrollShift=04H;
 fineScrollMask=0FH;
 vrtclPos=01FFH;
 vrtclPosShift=07H;
 horizPos=07FH;
 dftchMask=0FFH;
 vposrlof=08000H;

 ringtrigger=01H;
```

```
 1  anfracsize=06H;
 2  animhalf=020H;
 3
 4  b2Norm=0;
 5  b2Swap=1;
 6  b2Bobber=2;
 7
 8 TYPE
 9  VSpriteFlags=(vsprite,saveBack,overlay,mustDraw,vf4,vf5,vf6,v
10   bobUpdate,gelGone,vsOverflow);
11  VSpriteFlagSet=SET OF VSpriteFlags;
12  VSprite=RECORD
13   nextVSprite:VSpritePtr;
14   prevVSprite:VSpritePtr;
15   drawPath:VSpritePtr;
16   clearPath:VSpritePtr;
17   oldY:INTEGER;
18   oldX:INTEGER;
19   flags:VSpriteFlagSet;
20   y:INTEGER;
21   x:INTEGER;
22   height:INTEGER;
23   width:INTEGER;
24   depth:INTEGER;
25   meMask:BITSET;
26   hitMask:BITSET;
27   imageData:ADDRESS;
28   borderLine:ADDRESS;
29   collMask:ADDRESS;
30   sprColors:ADDRESS;
31   vsBob:BobPtr;
32   planePick:UByte;
33   planeOnOff:UByte;
34  END;
35  BobFlags=(
36   saveBob,bobIsComp,bf2,bf3,bf4,bf5,bf6,bf7,
37   bWaiting,bDrawn,bobsAway,bobNix,savePreserve,outStep
38  );
39  BobFlagSet=SET OF BobFlags;
40  Bob=RECORD
41   flags:BobFlagSet;
42   saveBuffer:ADDRESS;
43   imageShadow:ADDRESS;
44   before:BobPtr;
45   after:BobPtr;
46   bobVSprite:VSpritePtr;
47   bobComp:AnimCompPtr;
48   dBuffer:DBufPacketPtr;
49  END;
50  AnimComp=RECORD
51   flags:INTEGER;
52   timer:INTEGER;
53   timeSet:INTEGER;
54   nextComp:AnimCompPtr;
55   prevComp:AnimCompPtr;
56   nextSeq:AnimCompPtr;
57   prevSeq:AnimCompPtr;
```

```
  animCRoutine:ADDRESS;
  yTrans:INTEGER;
  xTrans:INTEGER;
  headOb:AnimObPtr;
  animBob:BobPtr;
END;
AnimOb=RECORD
  nextOb:AnimObPtr;
  prevOb:AnimObPtr;
  clock:LONGINT;
  anOldY:INTEGER;
  anOldX:INTEGER;
  anY:INTEGER;
  anX:INTEGER;
  yVel:INTEGER;
  xVel:INTEGER;
  yAccel:INTEGER;
  xAccel:INTEGER;
  ringYTrans:INTEGER;
  ringXTrans:INTEGER;
  animORoutine:ADDRESS;
  headComp:AnimCompPtr;
END;
DBufPacket=RECORD
  bufY:INTEGER;
  bufX:INTEGER;
  bufPath:VSpritePtr;
  bufBuffer:ADDRESS;
END;
CollTable=RECORD
  collPtrs:ARRAY [0..15] OF ADDRESS
END;
BitMap=RECORD
  bytesPerRow:CARDINAL;
  rows:CARDINAL;
  flags:UByte;
  depth:UByte;
  pad:CARDINAL;
  planes:ARRAY [0..7] OF ADDRESS;
END;
DisplayFlags=(ntsc,genloc,pal);
DisplayFlagSet=SET OF DisplayFlags;
GfxBase=RECORD
  libNode:Library;
  actiView:ViewPtr;
  copinit:CopinitPtr;
  cia:ADDRESS;
  blitter:ADDRESS;
  loFlist:ADDRESS;
  shFlist:ADDRESS;
  blthd:bltnodePtr;
  blttl:bltnodePtr;
  bsblthd:bltnodePtr;
  bsblttl:bltnodePtr;
  vbsrv:Interrupt;
  timsrv:Interrupt;
  bltsrv:Interrupt;
```

```
  textFonts:List;
  defaultFont:TextFontPtr;
  modes:BITSET;
  vBlank:UByte;
  debug:BYTE;
  beamSync:INTEGER;
  bplcon0:BITSET;
  spriteReserved:UByte;
  bytereserved:BYTE;
  flags:BITSET;
  blitLock:INTEGER;
  blitNest:INTEGER;
  blitWaitQ:List;
  blitOwner:TaskPtr;
  waitQ:List;
  displayFlags:DisplayFlagSet;
  simpleSprites:ADDRESS;
  maxDisplayRow:CARDINAL;
  maxDisplayColumn:CARDINAL;
  normalDisplayRows:CARDINAL;
  normalDisplayColumns:CARDINAL;
  normalDPMX:CARDINAL;
  normalDPMY:CARDINAL;
  lastChanceMemory:SignalSemaphorePtr;
  lcMptr:ADDRESS;
  microsPerLine:CARDINAL;
  reserved:ARRAY [0..1] OF LONGCARD;
 END;
 GfxBasePtr=POINTER TO GfxBase;

CONST
 blitMsgFault=4;

TYPE
 Isrvstr=RECORD
  node:Node;
  iptr:IsrvstrPtr;
  code:ADDRESS;
  ccode:ADDRESS;
  carg:INTEGER;
 END;
 LayerFlags=(
  layerSimple,layerSmart,layerSuper,lf3,layerUpdating,lf5,
  layerBackdrop,layerRefresh,layerClipRectsLost
 );
 LayerFlagSet=SET OF LayerFlags;
 LayerInfo=RECORD
  layer:LayerPtr;
  lp:LayerPtr;
  obs:LayerPtr;
  freeClipRects:MinList;
  lock:SignalSemaphore;
  head:List;
  longreserved:LONGINT;
  flags:LayerFlagSet;
  count:UByte;
  lockLayersCount:UByte;
```

```
  layerInfoExtraSize:CARDINAL;
  blitbuff:ADDRESS;
  layerInfoExtra:ADDRESS;
 END;

CONST
 lmnRegion=-1;
 newLayerInfoCalled=01H;
 alertLayersNoMem=083010000H;

TYPE
 AreaInfo=RECORD
  vctrTbl:ADDRESS;
  vctrPtr:ADDRESS;
  flagTbl:ADDRESS;
  flagPtr:ADDRESS;
  count:INTEGER;
  maxCount:INTEGER;
  firstX:INTEGER;
  firstY:INTEGER;
 END;
 TmpRas=RECORD
  rasPtr:ADDRESS;
  size:LONGINT;
 END;
 GelsInfo=RECORD
  sprRsrvd:BYTE;
  flags:BYTE;
  gelHead:VSpritePtr;
  gelTail:VSpritePtr;
  nextLine:ADDRESS;
  lastColor:ADDRESS;
  collHandler:CollTablePtr;
  leftmost:INTEGER;
  rightmost:INTEGER;
  topmost:INTEGER;
  bottommost:INTEGER;
  firstBlissObj:ADDRESS;
  lastBlissObj:ADDRESS;
 END;
 DrawModes=(dm0,complement,inversvid);
 DrawModeSet=SET OF DrawModes;
 FontStyles=(underlined,bold,italic,extended);
 FontStyleSet=SET OF FontStyles;
 FontFlags=(
  romFont,diskFont,revPath,tallDot,wideDot,proportional,
  designed,removed
 );
 FontFlagSet=SET OF FontFlags;
 RastPortFlags=(
 firstDot,oneDot,dBuffer,areaOutline,rpf4,noCrossFill,
  rpf6,rpf7,rpf8
 );
 RastPortFlagSet=SET OF RastPortFlags;
 RastPort=RECORD
  layer:LayerPtr;
  bitMap:BitMapPtr;
```

```
  areaPtrn:ADDRESS;
  tmpRas:TmpRasPtr;
  areaInfo:AreaInfoPtr;
  gelsInfo:GelsInfoPtr;
  mask:UByte;
  fgPen:UByte;
  bgPen:UByte;
  aOlPen:UByte;
  drawMode:DrawModeSet;
  areaPtSz:UByte;
  linPatCnt:UByte;
  dummy:BYTE;
  flags:RastPortFlagSet;
  linePtrn:CARDINAL;
  x:INTEGER;
  y:INTEGER;
  minterms:ARRAY [0..7] OF BYTE;
  penWidth:INTEGER;
  penHeight:INTEGER;
  font:TextFontPtr;
  algoStyle:FontStyleSet;
  txFlags:FontFlagSet;
  txHeight:CARDINAL;
  txWidth:CARDINAL;
  txBaseline:CARDINAL;
  txSpacing:INTEGER;
  user:ADDRESS;
  longreserved:ARRAY [0..1] OF LONGINT;
  wordreserved:ARRAY [0..6] OF WORD;
  reserved:ARRAY [0..7] OF BYTE;
 END;

CONST
 jam1=DrawModeSet{};
 jam2=DrawModeSet{dm0};
 spriteAttached=080H;
 normalFont=FontStyleSet{};

TYPE
 RegionRectangle=RECORD
  next:RegionRectanglePtr;
  prev:RegionRectanglePtr;
  bounds:Rectangle;
 END;
 Region=RECORD
  bounds:Rectangle;
  regionRectangle:RegionRectanglePtr;
 END;
 SimpleSprite=RECORD
  posctldata:ADDRESS;
  height:CARDINAL;
  x:CARDINAL;
  y:CARDINAL;
  num:CARDINAL;
 END;
 TextAttr=RECORD
  name:ADDRESS;
```

```
 ySize:CARDINAL;
 style:FontStyleSet;
 flags:FontFlagSet;
END;
TextFont=RECORD
 message:Message;
 ySize:CARDINAL;
 style:FontStyleSet;
 flags:FontFlagSet;
 xSize:CARDINAL;
 baseline:CARDINAL;
 boldSmear:CARDINAL;
 accessors:CARDINAL;
 loChar:CHAR;
 hiChar:CHAR;
 charData:ADDRESS;
 modulo:CARDINAL;
 charLoc:ADDRESS;
 charSpace:ADDRESS;
 charKern:ADDRESS;
END;
ColorMap=RECORD
 flags:UByte;
 type:UByte ;
 count:CARDINAL;
 colorTable:ADDRESS;
END;
ViewModes=(
 vm0,genlocVideo,lace,vm3,vm4,vm5,pfba,extraHalfbrite,
 genlocAudio,vm9,dualpf,ham,vm12,vpHide,sprites,hires
);
ViewModeSet=SET OF ViewModes;
ViewPort=RECORD
 next:ViewPortPtr;
 colorMap:ColorMapPtr;
 dspIns:CopListPtr;
 sprIns:CopListPtr;
 clrIns:CopListPtr;
 uCopIns:UCopListPtr;
 dWidth:INTEGER;
 dHeight:INTEGER;
 dxOffset:INTEGER;
 dyOffset:INTEGER;
 modes:ViewModeSet;
 reserved:CARDINAL;
 rasInfo:RasInfoPtr;
END;
View=RECORD
 viewPort:ViewPortPtr;
 lofCprList:CprlistPtr;
 shfCprList:CprlistPtr;
 dyOffset:INTEGER;
 dxOffset:INTEGER;
 modes:ViewModeSet;
END;
RasInfo=RECORD
 next:RasInfoPtr;
```

```
  bitMap:BitMapPtr;
  rxOffset:INTEGER;
  ryOffset:INTEGER;
 END;

(*
 * Die Prozeduren InitArea, InitBitMap, InitRastPort,
 * InitTmpRas, InitView und InitVPort haben einen
 * VAR Parameter.
 *)

PROCEDURE AddAnimOb(anOb{8}:AnimObPtr;
                    anKey{9}:ADDRESS;
                    rp{10}:RastPortPtr); CODE -156;
PROCEDURE AddBob(Bob{8}:BobPtr;
                 rp{9}:RastPortPtr); CODE -96;
PROCEDURE AddFont(textFont{9}:TextFontPtr); CODE -480;
PROCEDURE AddVSprite(vs{8}:VSpritePtr;
                     rp{9}:RastPortPtr); CODE -102;
PROCEDURE AllocRaster(width{0}:CARDINAL;
                      height{1}:CARDINAL):ADDRESS; CODE -492;
PROCEDURE AndRectRegion(region{8}:RegionPtr;
                        rectangle{9}:RectanglePtr); CODE -504;
PROCEDURE AndRegionRegion(
            region1{8}:RegionPtr;
            region2{9}:RegionPtr):BOOLEAN; CODE -624;
PROCEDURE Animate(
            anKey{8}:ADDRESS; rp{9}:RastPortPtr); CODE -162;
PROCEDURE AreaDraw(rp{9}:RastPortPtr;
                   x{0}:INTEGER;
                   y{1}:INTEGER):LONGINT; CODE -258;
PROCEDURE AreaEllipse(rp{9}:RastPortPtr;
                      cX{0}:INTEGER;
                      cY{1}:INTEGER;
                      a{2}:INTEGER;
                      b{3}:INTEGER):LONGINT; CODE -186;
PROCEDURE AreaEnd(rp{9}:RastPortPtr):LONGINT; CODE -264;
PROCEDURE AreaMove(rp{9}:RastPortPtr;
                   x{0}:INTEGER;
                   y{1}:INTEGER):LONGINT; CODE -252;
PROCEDURE AskFont(rp{9}:RastPortPtr;
                  textAttr{8}:TextAttrPtr); CODE -474;
PROCEDURE AskSoftStyle(
            rp{9}:RastPortPtr):FontStyleSet; CODE -84;
PROCEDURE AttemptLockLayerRom(
            layer{13}:LayerPtr):BOOLEAN; CODE -654;
PROCEDURE BltBitMap(srcBitMap{8}:BitMapPtr;
                    srcX{0}:INTEGER;
                    srcY{1}:INTEGER;
                    dstBitMap{9}:BitMapPtr;
                    dstX{2}:INTEGER;
                    dstY{3}:INTEGER;
                    sizeX{4}:INTEGER;
                    sizeY{5}:INTEGER;
                    minterm{6}:BYTE;
                    mask{7}:BYTE;
                    tempA{10}:ADDRESS):LONGCARD; CODE -30;
```

```
1  PROCEDURE BltBitMapRastPort(srcbm{8}:BitMapPtr;
2                              srcX{0}:INTEGER;
3                              srcY{1}:INTEGER;
4                              destRp{9}:RastPortPtr;
5                              destX{2}:INTEGER;
6                              destY{3}:INTEGER;
7                              sizeX{4}:INTEGER;
8                              sizeY{5}:INTEGER;
9                              minterm{6}:BYTE); CODE -606;
10 PROCEDURE BltClear(memBlock{9}:ADDRESS;
11                    bytecount{0}:LONGCARD;
12                    flags{1}:LONGCARD); CODE -300;
13 PROCEDURE BltMaskBitMapRastPort(
14             srcbm{8}:BitMapPtr;
15             srcX{0}:INTEGER;
16             srcY{1}:INTEGER;
17             destRp{9}:RastPortPtr;
18             destX{2}:INTEGER;
19             destY{3}:INTEGER;
20             sizeX{4}:INTEGER;
21             sizeY{5}:INTEGER;
22             minterm{6}:BYTE;
23             bltmask{10}:ADDRESS); CODE -636;
24 PROCEDURE BltPattern(rp{9}:RastPortPtr;
25                      mask{8}:ADDRESS;
26                      xl{0}:INTEGER;
27                      yl{1}:INTEGER;
28                      maxX{2}:INTEGER;
29                      maxY{3}:INTEGER;
30                      bytecnt{4}:INTEGER); CODE -312;
31 PROCEDURE BltTemplate(srcTemplate{8}:ADDRESS;
32                       srcX{0}:INTEGER;
33                       srcMod{1}:INTEGER;
34                       rp{9}:RastPortPtr;
35                       dstX{2}:INTEGER;
36                       dstY{3}:INTEGER;
37                       sizeX{4}:INTEGER;
38                       sizeY{5}:INTEGER); CODE -36;
39 PROCEDURE CBump(c{9}:UCopListPtr); CODE -366;
40 PROCEDURE ChangeSprite(vp{8}:ViewPortPtr;
41                        s{9}:SimpleSpritePtr;
42                        newdata{10}:ADDRESS); CODE -420;
43 PROCEDURE ClearEOL(rp{9}:RastPortPtr); CODE -42;
44 PROCEDURE ClearRectRegion(
45             region{8}:RegionPtr;
46             rectangle{9}:RectanglePtr):LONGINT; CODE -522;
47 PROCEDURE ClearRegion(region{8}:RegionPtr); CODE -528;
48 PROCEDURE ClearScreen(rp{9}:RastPortPtr); CODE -48;
49 PROCEDURE ClipBlit(src{8}:RastPortPtr;
50                    srcX{0}:INTEGER;
51                    srcY{1}:INTEGER;
52                    dest{9}:RastPortPtr;
53                    destX{2}:INTEGER;
54                    destY{3}:INTEGER;
55                    xSize{4}:INTEGER;
56                    ySize{5}:INTEGER;
57                    minterm{6}:BYTE); CODE -552;
```

```
1 PROCEDURE CloseFont(font{9}:TextFontPtr); CODE -78;
2 PROCEDURE CMove(c{9}:UCopListPtr;
3                 a{0}:ADDRESS;
4                 v{1}:INTEGER); CODE -372;
5 PROCEDURE CopySBitMap(layer{8}:LayerPtr); CODE -450;
6 PROCEDURE CWait(c{9}:UCopListPtr;
7                 v{0}:INTEGER;
8                 h{1}:INTEGER); CODE -378;
9 PROCEDURE DisownBlitter(); CODE -462;
10 PROCEDURE DisposeRegion(region{8}:RegionPtr); CODE -534;
11 PROCEDURE DoCollision(rp{9}:RastPortPtr); CODE -108;
12 PROCEDURE Draw(rp{9}:RastPortPtr;
13               x{0}:INTEGER;
14               y{1}:INTEGER); CODE -246;
15 PROCEDURE DrawEllipse(rp{9}:RastPortPtr;
16                       cX{0}:INTEGER;
17                       cY{1}:INTEGER;
18                       a{2}:INTEGER;
19                       b{3}:INTEGER); CODE -180;
20 PROCEDURE DrawGList(rp{9}:RastPortPtr;
21                     vp{8}:ViewPortPtr); CODE -114;
22 PROCEDURE Flood(rp{9}:RastPortPtr;
23                 mode{2}:LONGCARD;
24                 x{0}:INTEGER;
25                 y{1}:INTEGER):LONGINT; CODE -330;
26 PROCEDURE FreeColorMap(colorMap{8}:ColorMapPtr); CODE -576;
27 PROCEDURE FreeCopList(coplist{8}:CopListPtr); CODE -546;
28 PROCEDURE FreeCprList(cprlist{8}:CprlistPtr); CODE -564;
29 PROCEDURE FreeGBuffers(anOb{8}:AnimObPtr;
30                         rp{9}:RastPortPtr;
31                         db{0}:BOOLEAN); CODE -600;
32 PROCEDURE FreeRaster(p{8}:ADDRESS;
33                       width{0}:CARDINAL;
34                       height{1}:CARDINAL); CODE -498;
35 PROCEDURE FreeSprite(pick{0}:INTEGER); CODE -414;
36 PROCEDURE FreeVPortCopLists(vp{8}:ViewPortPtr); CODE -540;
37 PROCEDURE GetColorMap(
38              entries{0}:LONGINT):ColorMapPtr; CODE -570;
39 PROCEDURE GetGBuffers(anOb{8}:AnimObPtr;
40                       rp{9}:RastPortPtr;
41                       db{0}:BOOLEAN):BOOLEAN; CODE -168;
42 PROCEDURE GetRGB4(colorMap{8}:ColorMapPtr;
43                   entry{0}:LONGINT):LONGCARD; CODE -582;
44 PROCEDURE GetSprite(sprite{8}:SimpleSpritePtr;
45                     pick{0}:INTEGER):INTEGER; CODE -408;
46 (*PRIVATE*) PROCEDURE GraphicsReserved1():LONGINT; CODE -642;
47 (*PRIVATE*) PROCEDURE GraphicsReserved2():LONGINT; CODE -648;
48 PROCEDURE InitArea(VAR areainfo{8}:AreaInfo;
49                     buffer{9}:ADDRESS;
50                     maxvectors{0}:INTEGER); CODE -282;
51 PROCEDURE InitBitMap(VAR bm{8}:BitMap;
52                       depth{0}:INTEGER;
53                      width{1}:INTEGER;
54                       height{2}:INTEGER); CODE -390;
55 PROCEDURE InitGels(head{8}:VSpritePtr;
56                     tail{9}:VSpritePtr;
57                     gInfo{10}:GelsInfoPtr); CODE -120;
```

```
1 PROCEDURE InitGMasks(anOb{8}:AnimObPtr); CODE -174;
2 PROCEDURE InitMasks(vs{8}:VSpritePtr); CODE -126;
3 PROCEDURE InitRastPort(VAR rp{9}:RastPort); CODE -198;
4 PROCEDURE InitTmpRas(VAR tmpras{8}:TmpRas;
5                       buffer{9}:ADDRESS;
6                       size{0}:LONGINT); CODE -468;
7 PROCEDURE InitView(VAR view{9}:View); CODE -360;
8 PROCEDURE InitVPort(VAR vp{8}:ViewPort); CODE -204;
9 PROCEDURE LoadRGB4(vp{8}:ViewPortPtr;
10                     colors{9}:ADDRESS;
11                     count{0}:INTEGER); CODE -192;
12 PROCEDURE LoadView(view{9}:ViewPtr); CODE -222;
13 PROCEDURE LockLayerRom(layer{13}:LayerPtr); CODE -432;
14 PROCEDURE MakeVPort(view{8}:ViewPtr;
15                      viewport{9}:ViewPortPtr); CODE -216;
16 PROCEDURE Move(rp{9}:RastPortPtr;
17               x{0}:INTEGER;
18               y{1}:INTEGER); CODE -240;
19 PROCEDURE MoveSprite(vp{8}:ViewPortPtr;
20                       sprite{9}:SimpleSpritePtr;
21                       x{0}:INTEGER;
22                       y{1}:INTEGER); CODE -426;
23 PROCEDURE MrgCop(view{9}:ViewPtr); CODE -210;
24 PROCEDURE NewRegion():RegionPtr; CODE -516;
25 PROCEDURE OpenFont(
26             textAttr{8}:TextAttrPtr):TextFontPtr; CODE -72;
27 PROCEDURE OrRectRegion(
28             region{8}:RegionPtr;
29             rectangle{9}:RectanglePtr):LONGINT; CODE -510;
30 PROCEDURE OrRegionRegion(
31             region1{8}:RegionPtr;
32             region2{9}:RegionPtr):LONGINT; CODE -612;
33 PROCEDURE OwnBlitter(); CODE -456;
34 PROCEDURE PolyDraw(rp{9}:RastPortPtr;
35                     count{0}:INTEGER;
36                     array{8}:ADDRESS); CODE -336;
37 PROCEDURE QBlit(bp{9}:bltnodePtr); CODE -276;
38 PROCEDURE QBSBlit(bsp{9}:bltnodePtr); CODE -294;
39 PROCEDURE ReadPixel(rp{9}:RastPortPtr;
40                      x{0}:INTEGER;
41                      y{1}:INTEGER):LONGINT; CODE -318;
42 PROCEDURE RectFill(rp{9}:RastPortPtr;
43                     xMin{0}:INTEGER;
44                     yMin{1}:INTEGER;
45                     xMax{2}:INTEGER;
46                     yMax{3}:INTEGER); CODE -306;
47 PROCEDURE RemFont(textFont{9}:TextFontPtr); CODE -486;
48 PROCEDURE RemIBob(bob{8}:BobPtr;
49                    rp{9}:RastPortPtr;
50                    vp{10}:ViewPortPtr); CODE -132;
51 PROCEDURE RemVSprite(vs{8}:VSpritePtr); CODE -138;
52 PROCEDURE ScrollRaster(rp{9}:RastPortPtr;
53                         dx{0}:INTEGER;
54                         dy{1}:INTEGER;
55                         xMin{2}:INTEGER;
56                         yMin{3}:INTEGER;
57                         xMax{4}:INTEGER;
```

```
                        yMax{5}:INTEGER); CODE -396;
PROCEDURE ScrollVPort(vp{8}:ViewPortPtr); CODE -588;
PROCEDURE SetAPen(rp{9}:RastPortPtr;
                  pen{0}:CARDINAL); CODE -342;
PROCEDURE SetBPen(rp{9}:RastPortPtr;
                  pen{0}:CARDINAL); CODE -348;
PROCEDURE SetCollision(num{0}:LONGCARD;
                       routine{8}:PROC;
                       gInfo{9}:GelsInfoPtr); CODE -144;
PROCEDURE SetDrMd(rp{9}:RastPortPtr;
                  mode{0}:DrawModeSet); CODE -354;
PROCEDURE SetFont(rp{9}:RastPortPtr;
                  font{8}:TextFontPtr); CODE -66;
PROCEDURE SetRast(rp{9}:RastPortPtr;
                  pen{0}:CARDINAL); CODE -234;
PROCEDURE SetRGB4(vp{8}:ViewPortPtr;
                  n{0}:CARDINAL;
                  r{1}:CARDINAL;
                  g{2}:CARDINAL;
                  b{3}:CARDINAL); CODE -288;
PROCEDURE SetRGB4CM(cm{8}:ColorMapPtr;
                    n{0}:CARDINAL;
                    r{1}:CARDINAL;
                    g{2}:CARDINAL;
                    b{3}:CARDINAL); CODE -630;
PROCEDURE SetSoftStyle(
           rp{9}:RastPortPtr;
           style{0}:FontStyleSet;
           enable{1}:FontStyleSet):FontStyleSet; CODE -90;
PROCEDURE SortGList(rp{9}:RastPortPtr); CODE -150;
PROCEDURE SyncSBitMap(layer{8}:LayerPtr); CODE -444;
PROCEDURE Text(rp{9}:RastPortPtr;
               string{8}:ADDRESS;
               count{0}:INTEGER); CODE -60;
PROCEDURE TextLength(rp{9}:RastPortPtr;
                     string{8}:ADDRESS;
                     count{0}:INTEGER):INTEGER; CODE -54;
PROCEDURE UCopperListInit(copperList{8}:UCopListPtr;
                          num{0}:LONGINT); CODE -594;

PROCEDURE UnlockLayerRom(layer{13}:LayerPtr); CODE -438;
PROCEDURE VBeamPos():LONGINT; CODE -384;
PROCEDURE WaitBlit(); CODE -228;
PROCEDURE WaitBOVP(vp{8}:ViewPortPtr); CODE -402;
PROCEDURE WaitTOF(); CODE -270;
PROCEDURE WritePixel(rp{9}:RastPortPtr;
                     x{0}:INTEGER;
                     y{1}:INTEGER):LONGINT; CODE -324;
PROCEDURE XorRectRegion(
           region{8}:RegionPtr;
           rectangle{9}:RectanglePtr):LONGINT; CODE -558;
PROCEDURE XorRegionRegion(
           region1{8}:RegionPtr;
           region2{9}:RegionPtr):LONGINT; CODE -618;

END Graphics.

```

Hardware

```
(*$M-*)
DEFINITION MODULE Hardware;

FROM SYSTEM IMPORT
 ADDRESS,BITSET;

TYPE
 Byte=[-128..127];
 UByte=[0..255];

TYPE
 AdkFlags=(
  use0v1,use1v2,use2v3,use3vn,use0p1,use1p2,use2p3,use3pn,
  fast,msbSync,wordSync,uartBrk,mfmPrec,preComp0,preComp1,
  adkSet
 );
 AdkFlagSet=SET OF AdkFlags;

CONST
 pre000ns=AdkFlagSet{};
 pre140ns=AdkFlagSet{preComp0};
 pre280ns=AdkFlagSet{preComp1};
 pre560ns=AdkFlagSet{preComp0,preComp1};

TYPE
 DmaFlags=(
  aud0,aud1,aud2,aud3,disk,sprite,blitter,copper,
  raster,master,blithog,df11,df12,bltnzero,bltdone,dmaSet
 );
 DmaFlagSet=SET OF DmaFlags;

CONST
 dmaAll=DmaFlagSet{aud0..raster};

TYPE
 IntFlags=(
  tbe,dskblk,softint,ports,coper,vertb,blit,aud0i,
  aud1i,aud2i,aud3i,rbf,disksync,exter,inten,intSet
 );
 IntFlagSet=SET OF IntFlags;

CONST
 hSizeBits=6;
 vSizeBits=16-hSizeBits;
 hSizeMask=03FH;
 vSizeMask=03FFH;
 maxBytesPerRow=128;

TYPE
 BC0Flags=(
  nanbnc,nanbc,nabnc,nabc,anbnc,anbc,abnc,abc,
  dest,srcC,srcB,srcA,ash1,ash2,ash4,ash8
 );
```

```
 1  BC0FlagSet=SET OF BC0Flags;
 2
 3 CONST
 4  aORb=BC0FlagSet{abc,anbc,nabc,abnc,anbnc,nabnc};
 5  aORc=BC0FlagSet{abc,nabc,abnc,anbc,nanbc,anbnc};
 6  aXORc=BC0FlagSet{nabc,abnc,nanbc,anbnc};
 7  aTOd=BC0FlagSet{abc,anbc,abnc,anbnc};
 8
 9 TYPE
10  BC1Flags=(
11   lineMode,desc,fillCarryIn,fillOr,fillXor,ovFlag,signFlag,
12   bf7,bf8,bf9,bf10,bf11,bsh1,bsh2,bsh4,bsh8
13  );
14  BC1FlagSet=SET OF BC1Flags;
15 CONST
16  oneDot=desc;
17  blitReverse=desc;
18  aul=BC1FlagSet{fillCarryIn};
19  sul=BC1FlagSet{fillOr};
20  sud=BC1FlagSet{fillXor};
21  octant1=sud;
22  octant2=BC1FlagSet{};
23  octant3=sul;
24  octant4=aul+sud;
25  octant5=aul+sul+sud;
26  octant6=aul+sul;
27  octant7=aul;
28  octant8=sul+sud;
29
30 TYPE
31  BltnodePtr=POINTER TO Bltnode;
32  Bltnode=RECORD
33   n:BltnodePtr;
34   function:ADDRESS;
35   stat:CHAR;
36   blitsize:INTEGER;
37   beamsync:INTEGER;
38   cleanup:ADDRESS;
39  END;
40
41 CONST
42  cleanup=40H;
43  cleanme=cleanup;
44
45 TYPE
46  Coord=RECORD
47   v,h:Byte;
48  END;
49  Custom=RECORD
50   bltddat:CARDINAL;
51   dmaconr:DmaFlagSet;
52   vposr:LONGCARD;
53   dskdatr:CARDINAL;
54   joy0dat:Coord;
55   joy1dat:Coord;
56   clxdat:CARDINAL;
57   adkconr:BITSET;
```

```
1   pot0dat:CARDINAL;
2   pot1dat:CARDINAL;
3   potinp:CARDINAL;
4   serdatr:CARDINAL;
5   dskbytr:CARDINAL;
6   intenar:BITSET;
7   intreqr:CARDINAL;
8   dskpt:ADDRESS;
9   dsklen:CARDINAL;
10  dskdat:CARDINAL;
11  refptr:CARDINAL;
12  vposw:CARDINAL;
13  vhposw:CARDINAL;
14  copcon:BITSET;
15  serdat:CARDINAL;
16  serper:CARDINAL;
17  potgo:CARDINAL;
18  joytest:CARDINAL;
19  strequ:CARDINAL;
20  strvbl:CARDINAL;
21  strhor:CARDINAL;
22  strlong:CARDINAL;
23  bltcon0:BITSET;
24  bltcon1:BITSET;
25  bltafwm:CARDINAL;
26  bltalwm:CARDINAL;
27  bltcpt:ADDRESS;
28  bltbpt:ADDRESS;
29  bltapt:ADDRESS;
30  bltdpt:ADDRESS;
31  bltsize:CARDINAL;
32  pad2d:ARRAY [0..2] OF CARDINAL;
33  bltcmod:CARDINAL;
34  bltbmod:CARDINAL;
35  bltamod:CARDINAL;
36  bltdmod:CARDINAL;
37  pad34:ARRAY [0..3] OF CARDINAL;
38  bltcdat:CARDINAL;
39  bltbdat:CARDINAL;
40  bltadat:CARDINAL;
41  pad3b:ARRAY [0..3] OF CARDINAL;
42  dsksync:CARDINAL;
43  cop1lc:LONGCARD;
44  cop2lc:LONGCARD;
45  copjmp1:CARDINAL;
46  copjmp2:CARDINAL;
47  copins:CARDINAL;
48  diwstrt:CARDINAL;
49  diwstop:CARDINAL;
50  ddfstrt:CARDINAL;
51  ddfstop:CARDINAL;
52  dmacon:BITSET;
53  clxcon:CARDINAL;
54  intena:BITSET;
55  intreq:CARDINAL;
56  adkcon:BITSET;
57  aud:ARRAY [0..3] OF RECORD
```

```
    acptr:ADDRESS;
    aclen:CARDINAL;
    acper:CARDINAL;
    acvol:CARDINAL;
    acdat:CARDINAL;
    acpad:ARRAY [0..1] OF CARDINAL;
  END;
  bplpt:ARRAY [0..5] OF ADDRESS;
  pad7c:ARRAY [0..3] OF CARDINAL;
  bplcon0:CARDINAL;
  bplcon1:CARDINAL;
  bplcon2:CARDINAL;
  pad83:CARDINAL;
  bpl1mod:CARDINAL;
  bpl2mod:CARDINAL;
  pad86:ARRAY [0..1] OF CARDINAL;
  bpldat:ARRAY [0..5] OF CARDINAL;
  pad8e:ARRAY [0..1] OF CARDINAL;
  sprpt:ARRAY [0..7] OF ADDRESS;
  spr:ARRAY [0..7] OF RECORD
   pos:CARDINAL;
   ctl:CARDINAL;
   dataa:CARDINAL;
   datab:CARDINAL;
  END;
  color:ARRAY [0..31] OF CARDINAL;
 END;

VAR
 custom[0DFF000H]:Custom;

TYPE
 CiaIcrFlags=(ta,tb,alrm,sp,flg,if5,if6,setclr);
 CiaIcrFlagSet=SET OF CiaIcrFlags;
CONST
 ir=setclr; (* On read setclr has the meaning of ir *)
TYPE
 CiaCraFlags=(
  craStart,craPbon,craOutmode,craRunmode,craLoad,craInmode,
  craSpmode,craTodin
 );
 CiaCraFlagSet=SET OF CiaCraFlags;
 CiaCrbFlags=(
  crbStart,crbPbon,crbOutmode,crbRunmode,crbLoad,crbInmode0,
  crbInmode1,crbAlarm
 );
 CiaCrbFlagSet=SET OF CiaCrbFlags;
 CiaaPraFlags=(
  overlay,led,dskChange,dskProt,dskTrack0,dskRdy,gamePort0,
  gamePort1
 );
 CiaaPraFlagSet=SET OF CiaaPraFlags;
 CiaaPrbFlags=[0..7]; (* Parallel port *)
 CiaaPrbFlagSet=SET OF CiaaPrbFlags;
 CiabPraFlags=(
  prtrBusy,prtrPOut,prtrSel,comDSR,comCTS,comCD,comRTS,comDTR
 );
```

```
 1  CiabPraFlagSet=SET OF CiabPraFlags;
 2  CiabPrbFlags=(
 3   dskStep,dskDirec,dskSide,dskSel0,dskSel1,dskSel2,dskSel3,
 4   dskMotor
 5  );
 6  CiabPrbFlagSet=SET OF CiabPrbFlags;
 7
 8 TYPE
 9  ShortSet=SET OF [0..7];
10  Pad=ARRAY [0..253] OF ShortSet;
11  CIAA=RECORD
12   pra:CiaaPraFlagSet; pad0:Pad;
13   prb:CiaaPrbFlagSet; pad1:Pad;
14   ddra:CiaaPraFlagSet; pad2:Pad;
15   ddrb:CiaaPrbFlagSet; pad3:Pad;
16   talo:UByte; pad4:Pad;
17   tahi:UByte; pad5:Pad;
18   tblo:UByte; pad6:Pad;
19   tbhi:UByte; pad7:Pad;
20   todlow:UByte; pad8:Pad;
21   todmid:UByte; pad9:Pad;
22   todhi:UByte; pad10:Pad;
23   unusedreg:ShortSet; pad11:Pad;
24   sdr:ShortSet; pad12:Pad;
25   icr:CiaIcrFlagSet; pad13:Pad;
26   cra:CiaCraFlagSet; pad14:Pad;
27   crb:CiaCrbFlagSet;
28  END;
29  CIAB=RECORD
30   pra:CiabPraFlagSet; pad0:Pad;
31   prb:CiabPrbFlagSet; pad1:Pad;
32   ddra:CiabPraFlagSet; pad2:Pad;
33   ddrb:CiabPrbFlagSet; pad3:Pad;
34   talo:UByte; pad4:Pad;
35   tahi:UByte; pad5:Pad;
36   tblo:UByte; pad6:Pad;
37   tbhi:UByte; pad7:Pad;
38   todlow:UByte; pad8:Pad;
39   todmid:UByte; pad9:Pad;
40   todhi:UByte; pad10:Pad;
41   unusedreg:ShortSet; pad11:Pad;
42   sdr:ShortSet; pad12:Pad;
43   icr:CiaIcrFlagSet; pad13:Pad;
44   cra:CiaCraFlagSet; pad14:Pad;
45   crb:CiaCrbFlagSet;
46  END;
47
48 VAR
49  ciaa[0BFE001H]:CIAA;
50  ciab[0BFD000H]:CIAB;
51
52 END Hardware.
53
```

Icon

```
 5 DEFINITION MODULE Icon {"icon.library",33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS;
 9 FROM Workbench IMPORT
10  FreeListPtr,DiskObjectPtr;
11
12 PROCEDURE BumpRevision(new{8},old{9}:ADDRESS); CODE -108;
13
14 PROCEDURE AddFreeList(free{8}:FreeListPtr; mem{9}:ADDRESS;
15                       len{10}:LONGINT):LONGINT; CODE -72;
16 PROCEDURE FreeFreeList(free{8}:FreeListPtr); CODE -54;
17
18 PROCEDURE FindToolType(
19             toolTypes{8},
20             typeName{9}:ADDRESS):ADDRESS; CODE -96;
21 PROCEDURE MatchToolValue(
22             typeString{8},val{9}:ADDRESS):LONGINT; CODE -102;
23
24 PROCEDURE AllocWBObject():ADDRESS; CODE -66;
25 PROCEDURE GetWBObject(name{8}:ADDRESS):ADDRESS; CODE -30;
26 PROCEDURE PutWBObject(name{8}:ADDRESS;
27                       obj{9}:ADDRESS):LONGINT; CODE -36;
28 PROCEDURE FreeWBObject(obj{8}:ADDRESS); CODE -60;
29
30 PROCEDURE GetIcon(name{8}:ADDRESS; icon{9}:DiskObjectPtr;
31                   f{10}:FreeListPtr):LONGINT; CODE -42;
32 PROCEDURE PutIcon(name{8}:ADDRESS;
33                   obj{9}:DiskObjectPtr):LONGINT; CODE -48;
34
35 PROCEDURE GetDiskObject(
36             name{8}:ADDRESS):DiskObjectPtr; CODE -78;
37 PROCEDURE PutDiskObject(
38             name{8}:ADDRESS;
39             obj{9}:DiskObjectPtr):LONGINT; CODE -84;
40 PROCEDURE FreeDiskObject(obj{8}:DiskObjectPtr); CODE -90;
41
42 END Icon.
43
```

Input

```
 5 (* $M- *)
 6 DEFINITION MODULE Input;
 7
 8 FROM Exec IMPORT
 9  nonstd;
10
11 CONST
12  inputName="input.device";
13  addHandler=nonstd+0;
14  remHandler=nonstd+1;
15  writeEvent=nonstd+2;
16  setThresh=nonstd+3;
17  setPeriod=nonstd+4;
18  setMPort=nonstd+5;
19  setMType=nonstd+6;
20  setMTrig=nonstd+7;
21
22 END Input.
23
```

InputEvent

```
 5 (*$M-*)
 6 DEFINITION MODULE InputEvent;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Timer IMPORT
11  TimeVal;
12
13 TYPE
14  Class=(
15   null,rawkey,rawmouse,event,pointerpos,c15,timer,gadgetdown,
16   gadgetup,requester,menulist,closewindow,sizewindow,
17   refreshwindow,newprefs,diskremoved,diskinserted,
18   activewindow,inactivewindow
19  );
20
21 CONST
22  classMax=ORD(MAX(Class));
23  upPrefix=080H;
24  keyCodeFirst=00H;
25  keyCodeLast=077H;
26  commCodeFirst=078H;
27  commCodeLast=07FH;
28  c0First=00H;
29  c0Last=01FH;
30  asciiFirst=020H;
31  asciiLast=07EH;
32  asciiDel=07FH;
33  c1First=080H;
34  c1Last=09FH;
35  latin1First=0A0H;
36  latin1Last=0FFH;
37  lButton=068H;
38  rButton=069H;
39  mButton=06AH;
40  noButton=0FFH;
41  newActive=01H;
42  reqClear=00H;
43  reqSet=01H;
44
45 TYPE
46  Qualifiers=(
47   lShift,rShift,capsLock,control,lAlt,rAlt,lCommand,rCommand,
48   numericPad,repeat,interrupt,multiBroadcast,midButton,
49   rightButton,leftButton,relativeMouse
50  );
51  QualifierSet=SET OF Qualifiers;
52  InputEventPtr=POINTER TO InputEvent;
53  InputEvent=RECORD
54   nextEvent:InputEventPtr;
55   class:Class;
56   subClass:Class;
57   code:CARDINAL;
```

```
1    qualifier:QualifierSet;
2    CASE :INTEGER OF
3    |0: x,y:INTEGER;
4    |1: eventAddress:ADDRESS
5    END;
6    timeStamp:TimeVal
7   END;
8
9  END InputEvent.
10
```

Intuition

```
 5 DEFINITION MODULE Intuition {"intuition.library",33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS,BITSET,BYTE,CAST,LONGSET;
 9 FROM Exec IMPORT
10  Byte,Interrupt,IOStdReq,Library,List,MemReqSet,Message,
11  MsgPortPtr,SignalSemaphore,UByte;
12 FROM Graphics IMPORT
13  jam2,BitMap,BitMapPtr,ClipRect,DrawModeSet,GfxBasePtr,
14  LayerInfo,LayerPtr,RastPort,RastPortPtr,RegionPtr,
15  SimpleSpritePtr,TextAttr,TextAttrPtr,TextFontPtr,TmpRas,
16  View,ViewModeSet,ViewPort,ViewPortPtr,ViewPtr;
17 FROM Timer IMPORT
18  TimeRequest,TimeVal;
19 FROM InputEvent IMPORT
20  lButton,rButton,upPrefix,InputEvent,InputEventPtr,
21  Qualifiers,QualifierSet;
22 FROM KeyMap IMPORT
23  KeyMapPtr;
24
25 TYPE
26  BorderPtr=POINTER TO Border;
27  GadgetPtr=POINTER TO Gadget;
28  ImagePtr=POINTER TO Image;
29  IntuiMessagePtr=POINTER TO IntuiMessage;
30  IntuiTextPtr=POINTER TO IntuiText;
31  MenuItemPtr=POINTER TO MenuItem;
32  MenuPtr=POINTER TO Menu;
33  PreferencesPtr=POINTER TO Preferences;
34  PropInfoPtr=POINTER TO PropInfo;
35  RememberPtr=POINTER TO Remember;
36  RequesterPtr=POINTER TO Requester;
37  ScreenPtr=POINTER TO Screen;
38  StringInfoPtr=POINTER TO StringInfo;
39  WindowPtr=POINTER TO Window;
40
41 CONST
42  menuEnabled=0;
43  miDrawn=8;
44
45 TYPE
46  Menu=RECORD
47   nextMenu:MenuPtr;
48   leftEdge:INTEGER;
49   topEdge:INTEGER;
50   width:INTEGER;
51   height:INTEGER;
52   flags:BITSET;
53   menuName:ADDRESS;
54   firstItem:MenuItemPtr;
55   jazzX:INTEGER;
56   jazzY:INTEGER;
57   beatX:INTEGER;
```

```
 1   beatY:INTEGER;
 2  END;
 3  MenuItemFlags=(
 4   checkIt,itemText,commSeq,menuToggle,itemEnabled,mif5,
 5   highComp,highBox,checked,mif9,mif10,mif11,isDrawn,
 6   highItem,menuToggled
 7  );
 8  MenuItemFlagSet=SET OF MenuItemFlags;
 9
10 CONST
11  highNone=MenuItemFlagSet{highBox,highComp};
12  checkWidth=19;
13  commWidth=27;
14  lowCheckWidth=13;
15  lowCommWidth=16;
16
17 TYPE
18  MenuItem=RECORD
19   nextItem:MenuItemPtr;
20   leftEdge:INTEGER;
21   topEdge:INTEGER;
22   width:INTEGER;
23   height:INTEGER;
24   flags:MenuItemFlagSet;
25   mutualExclude:LONGSET;
26   itemFill:ADDRESS;
27   selectFill:ADDRESS;
28   command:CHAR;
29   subItem:MenuItemPtr;
30   nextSelect:CARDINAL;
31  END;
32  RequesterFlags=(
33   pointRel,preDrawn,noisyReq,rf3,rf4,rf5,rf6,rf7,rf8,rf9,
34   rf10,rf11,reqOffWindow,reqActive,sysRequest,deferRefresh
35  );
36  RequesterFlagSet=SET OF RequesterFlags;
37  Requester=RECORD
38   olderRequest:RequesterPtr;
39   leftEdge:INTEGER;
40   topEdge:INTEGER;
41   width:INTEGER;
42   height:INTEGER;
43   relLeft:INTEGER;
44   relTop:INTEGER;
45   reqGadget:GadgetPtr;
46   reqBorder:BorderPtr;
47   reqText:IntuiTextPtr;
48   flags:RequesterFlagSet;
49   backFill:UByte;
50   reqLayer:LayerPtr;
51   reqPad1:ARRAY [0..31] OF BYTE;
52   imageBMap:BitMapPtr;
53   rWindow:WindowPtr;
54   reqPad2:ARRAY [0..35] OF BYTE;
55  END;
56  GadgetFlags=(
57   gadgHBox,gadgHImage,gadgImage,gRelBottom,gRelRight,
```

```
  gRelWidth,gRelHeight,selected,gadgDisabled
 );
 GadgetFlagSet=SET OF GadgetFlags;
 ActivationFlags=(
  relVerify,gadgImmediate,endGadget,followMouse,rightBorder,
  leftBorder,topBorder,bottomBorder,toggleSelect,stringCenter,
  stringRight,longint,altKeyMap,boolExtend
 );
 ActivationFlagSet=SET OF ActivationFlags;

CONST
 gadgHighbits=CAST(GadgetFlagSet,03H);
 gadgHNone=GadgetFlagSet{gadgHBox,gadgHImage};
 boolGadget=0001H;
 gadget0002=0002H;
 propGadget=0003H;
 strGadget=0004H;
 sizing=0010H;
 wDragging=0020H;
 sDragging=0030H;
 wUpFront=0040H;
 sUpFront=0050H;
 wDownBack=0060H;
 sDownBack=0070H;
 close=0080H;
 reqGadget=1000H;
 gzzGadget=2000H;
 scrGadget=4000H;
 sysGadget=8000H;
 gadgetType=CAST(BITSET,0FC00H);

TYPE
 Gadget=RECORD
  nextGadget:GadgetPtr;
  leftEdge:INTEGER;
  topEdge:INTEGER;
  width:INTEGER;
  height:INTEGER;
  flags:GadgetFlagSet;
  activation:ActivationFlagSet;
  gadgetType:CARDINAL;
  gadgetRender:ADDRESS;
  selectRender:ADDRESS;
  gadgetText:IntuiTextPtr;
  mutualExclude:LONGSET;
  specialInfo:ADDRESS;
  gadgetID:INTEGER;
  userData:ADDRESS;
 END;

CONST
 boolMask=1;

TYPE
 BoolInfo=RECORD
  flags:BITSET;
  mask:ADDRESS;
```

```
 1    reserved:LONGCARD;
 2   END;
 3   PropInfoFlags=(
 4    autoKnob,freeHoriz,freeVert,propBorderless,pf4,pf5,pf6,pf7,
 5    knobHit
 6   );
 7   PropInfoFlagSet=SET OF PropInfoFlags;
 8
 9  CONST
10   knobVmin=4;
11   knobHmin=6;
12   maxBody=0FFFFH;
13   maxPot=0FFFFH;
14
15  TYPE
16   PropInfo=RECORD
17    flags:PropInfoFlagSet;
18    horizPot:CARDINAL;
19    vertPot:CARDINAL;
20    horizBody:CARDINAL;
21    vertBody:CARDINAL;
22    cWidth:CARDINAL;
23    cHeight:CARDINAL;
24    hPotRes:CARDINAL;
25    vPotRes:CARDINAL;
26    leftBorder:CARDINAL;
27    topBorder:CARDINAL;
28   END;
29
30   StringInfo=RECORD
31    buffer:ADDRESS;
32    undoBuffer:ADDRESS;
33    bufferPos:INTEGER;
34    maxChars:INTEGER;
35    dispPos:INTEGER;
36    undoPos:INTEGER;
37    numChars:INTEGER;
38    dispCount:INTEGER;
39    cLeft:INTEGER;
40    cTop:INTEGER;
41    layerPtr:LayerPtr;
42    longInt:LONGINT;
43    altKeyMap:KeyMapPtr;
44   END;
45
46  CONST
47   autoFrontPen=0;
48   autoBackPen=1;
49   autoDrawMode=jam2;
50   autoLeftEdge=6;
51   autoTopEdge=3;
52   autoITextFont=NIL;
53   autoNextText=NIL;
54  TYPE
55   IntuiText=RECORD
56    frontPen:UByte;
57    backPen:UByte;
```

```
  drawMode:DrawModeSet;
  leftEdge:INTEGER;
  topEdge:INTEGER;
  iTextFont:TextAttrPtr;
  iText:ADDRESS;
  nextText:IntuiTextPtr;
END;

Border=RECORD
  leftEdge:INTEGER;
  topEdge:INTEGER;
  frontPen:UByte;
  backPen:UByte;
  drawMode:DrawModeSet;
  count:UByte;
  xy:ADDRESS;
  nextBorder:BorderPtr;
END;

Image=RECORD
  leftEdge:INTEGER;
  topEdge:INTEGER;
  width:INTEGER;
  height:INTEGER;
  depth:INTEGER;
  imageData:ADDRESS;
  planePick:UByte;
  planeOnOff:UByte;
  nextImage:ImagePtr;
END;

IDCMPFlags=(
  sizeVerify,newSize,refreshWindow,mouseButtons,mouseMove,
  gadgetDown,gadgetUp,reqSet,menuPick,closeWindow,rawKey,
  reqVerify,reqClear,menuVerify,newPrefs,diskInserted,
  diskRemoved,wbenchMessage,activeWindow,inactiveWindow,
  deltaMove,vanillaKey,intuiTicks,c23,c24,c25,c26,c27,c28,
  c29,c30,lonelyMessage
);
IDCMPFlagSet=SET OF IDCMPFlags;

CONST
  selectUp=lButton+upPrefix;     (* mouseButtons *)
  selectDown=lButton;
  menuUp=rButton+upPrefix;
  menuDown=rButton;
  menuNull=0FFFFH;               (* menuPick *)
  noMenu=1FH; noItem=3FH; noSub=1FH;
  keyCodeQ=10H;                  (* rawKey *)
  keyCodeX=32H;
  keyCodeV=34H;
  keyCodeB=35H;
  keyCodeN=36H;
  keyCodeM=37H;
  cursorUp=4CH;
  cursorDown=4DH;
  cursorRight=4EH;
```

```
 1  cursorLeft=4FH;
 2  menuHot=1;                    (* menuVerify *)
 3  menuCancel=2;
 4  menuWaiting=3;
 5  okOk=menuHot;
 6  okAbort=4;
 7  okCancel=menuCancel;
 8  wbenchOpen=1;                    (* wbenchMessage *)
 9  wbenchClose=2;
10 (* IntuiMessage.qualifier *)
11  altLeft=QualifierSet{lAlt};
12  altRight=QualifierSet{rAlt};
13  amigaLeft=QualifierSet{lCommand};
14  amigaRight=QualifierSet{rCommand};
15  amigaKeys=amigaLeft+amigaRight;
16
17 TYPE
18  IntuiMessage=RECORD
19   execMessage:Message;
20   class:IDCMPFlagSet;
21   code:CARDINAL;
22   qualifier:QualifierSet;
23   iAddress:ADDRESS;
24   mouseX:INTEGER;
25   mouseY:INTEGER;
26   seconds:LONGCARD;
27   micros:LONGCARD;
28   idcmpWindow:WindowPtr;
29   specialLink:IntuiMessagePtr;
30  END;
31
32 TYPE
33  WindowFlags=(
34   windowSizing,windowDrag,windowDepth,windowClose,sizeBRight,
35   sizeBBottom,simpleRefresh,superBitMap,backDrop,reportMouse,
36   gimmeZeroZero,borderless,activate,windowActive,inRequest,
37   menuState,rmbTrap,noCareRefresh,wf18,wf19,wf20,wf21,wf22,
38   wf23,windowRefresh,wbenchWindow,windowTicked,wf27,wf28,
39   wf29,wf30,wf31
40  );
41  WindowFlagSet=SET OF WindowFlags;
42  ScreenFlags=(
43   wbenchScreen,sf1,sf2,sf3,showTitle,beeping,customBitMap,
44   screenBehind,screenQuiet);
45  ScreenFlagSet=SET OF ScreenFlags;
46
47 CONST
48  otherRefresh=WindowFlagSet{simpleRefresh,superBitMap};
49  superUnused=WindowFlagSet{wf18..wf23,wf27..wf31};
50  stdScreenHeight=-1;
51  customScreen=ScreenFlagSet{wbenchScreen..sf3};
52
53 TYPE
54  NewWindow=RECORD
55   leftEdge:INTEGER;
56   topEdge:INTEGER;
57   width:INTEGER;
```

```
  height:INTEGER;
  detailPen:Byte;
  blockPen:Byte;
  idcmpFlags:IDCMPFlagSet;
  flags:WindowFlagSet;
  firstGadget:GadgetPtr;
  checkMark:ImagePtr;
  title:ADDRESS;
  screen:ScreenPtr;
  bitMap:BitMapPtr;
  minWidth:INTEGER;
  minHeight:INTEGER;
  maxWidth:INTEGER;
  maxHeight:INTEGER;
  type:ScreenFlagSet;
END;
NewScreen=RECORD
  leftEdge:INTEGER;
  topEdge:INTEGER;
  width:INTEGER;
  height:INTEGER;
  depth:INTEGER;
  detailPen:Byte;
  blockPen:Byte;
  viewModes:ViewModeSet;
  type:ScreenFlagSet;
  font:TextAttrPtr;
  defaultTitle:ADDRESS;
  gadgets:GadgetPtr;
  customBitMap:BitMapPtr;
END;
Window=RECORD
  nextWindow:WindowPtr;
  leftEdge:INTEGER;
  topEdge:INTEGER;
  width:INTEGER;
  height:INTEGER;
  mouseY:INTEGER;
  mouseX:INTEGER;
  minWidth:INTEGER;
  minHeight:INTEGER;
  maxWidth:INTEGER;
  maxHeight:INTEGER;
  flags:WindowFlagSet;
  menuStrip:MenuPtr;
  title:ADDRESS;
  firstRequest:RequesterPtr;
  dmRequest:RequesterPtr;
  reqCount:INTEGER;
  wScreen:ScreenPtr;
  rPort:RastPortPtr;
  borderLeft:Byte;
  borderTop:Byte;
  borderRight:Byte;
  borderBottom:Byte;
  borderRPort:RastPortPtr;
  firstGadget:GadgetPtr;
```

```
  parent:WindowPtr;
  descendant:WindowPtr;
  pointer:ADDRESS;
  ptrHeight:Byte;
  ptrWidth:[0..16];
  xOffset:Byte;
  yOffset:Byte;
  idcmpFlags:IDCMPFlagSet;
  userPort:MsgPortPtr;
  windowPort:MsgPortPtr;
  messageKey:IntuiMessagePtr;
  detailPen:UByte;
  blockPen:UByte;
  checkMark:ImagePtr;
  screenTitle:ADDRESS;
  gzzMouseX:INTEGER;
  gzzMouseY:INTEGER;
  gzzWidth:INTEGER;
  gzzHeight:INTEGER;
  extData:ADDRESS;
  userData:ADDRESS;
  wLayer:LayerPtr;
  iFont:TextFontPtr;
END;
Screen=RECORD
  nextScreen:ScreenPtr;
  firstWindow:WindowPtr;
  leftEdge:INTEGER;
  topEdge:INTEGER;
  width:INTEGER;
  height:INTEGER;
  mouseY:INTEGER;
  mouseX:INTEGER;
  flags:ScreenFlagSet;
  title:ADDRESS;
  defaultTitle:ADDRESS;
  barHeight:Byte;
  barVBorder:Byte;
  barHBorder:Byte;
  menuVBorder:Byte;
  menuHBorder:Byte;
  wBorTop:Byte;
  wBorLeft:Byte;
  wBorRight:Byte;
  wBorBottom:Byte;
  font:TextAttrPtr;
  viewPort:ViewPort;
  rastPort:RastPort;
  bitMap:BitMap;
  layerInfo:LayerInfo;
  firstGadget:GadgetPtr;
  detailPen:UByte;
  blockPen:UByte;
  saveColor0:CARDINAL;
  barLayer:LayerPtr;
  extData:ADDRESS;
  userData:ADDRESS;
```

```
 1  END;
 2
 3 CONST
 4  filenameSize=30;
 5  pointerSize=(1+16+1)*2;
 6  topazEighty=8;
 7  topazSixty=9;
 8
 9 TYPE
10  PrinterPort=(parallelPrinter,serialPrinter);
11  PrinterType=(
12   customName,alphaP101,brother15XL,cbmMps1000,diab630,
13   diabAdvD25,diabC150,epson,epsonJX80,okimate20,QumeLP20,
14   hpLaserjet,hpLaserjetPlus
15  );
16  SerParShk=(
17   shakeXon,shakeRts,shakeNone,sps3,parityNone,parityEven,
18   parityOdd
19  );
20  SerParShkSet=SET OF SerParShk;
21  Preferences=RECORD
22   fontHeight:UByte;
23   printerPort:PrinterPort;
24   baudRate:CARDINAL;
25   keyRptSpeed:TimeVal;
26   keyRptDelay:TimeVal;
27   doubleClick:TimeVal;
28   pointerMatrix:ARRAY [0..pointerSize-1] OF CARDINAL;
29   xOffset:Byte;
30   yOffset:Byte;
31   color17:CARDINAL;
32   color18:CARDINAL;
33   color19:CARDINAL;
34   pointerTicks:CARDINAL;
35   color0:CARDINAL;
36   color1:CARDINAL;
37   color2:CARDINAL;
38   color3:CARDINAL;
39   viewXOffset:Byte;
40   viewYOffset:Byte;
41   viewInitX:INTEGER;
42   viewInitY:INTEGER;
43   enableCLI:BOOLEAN;
44   printerType:PrinterType;
45   printerFilename:ARRAY [0..filenameSize-1] OF CHAR;
46   printPitch:CARDINAL;
47   printQuality:CARDINAL;
48   printSpacing:CARDINAL;
49   printLeftMargin:CARDINAL;
50   printRightMargin:CARDINAL;
51   printImage:CARDINAL;
52   printAspect:CARDINAL;
53   printShade:CARDINAL;
54   printThreshold:INTEGER;
55   paperSize:CARDINAL;
56   paperLength:CARDINAL;
57   paperType:CARDINAL;
```

```
1    serRWBits:UByte;
2    serStopBuf:UByte;
3    serParShk:SerParShkSet;
4    laceWB:BOOLEAN;
5    workName:ARRAY [0..filenameSize-1] OF CHAR;
6    padding:ARRAY [0..15] OF BYTE;
7   END;
8  CONST
9   baud110=0;
10  baud300=1;
11  baud1200=2;
12  baud2400=3;
13  baud4800=4;
14  baud9600=5;
15  baud19200=6;
16  baudMidi=7;
17  pica=0H;
18  elite=0400H;
19  fine=0800H;
20  draft=0H;
21  letter=0100H;
22  sixLPI=0H;
23  eightLPI=0200H;
24  imagePositive=0;
25  imageNegative=1;
26  aspectHoriz=0;
27  aspectVert=1;
28  shadeBW=0;
29  shadeGreyscale=1;
30  shadeColor=2;
31  usLetter=0H;
32  usLegal=010H;
33  nTractor=020H;
34  wTractor=030H;
35  custom=040H;
36  fanfold=0H;
37  single=080H;
38  readBits=0F0H;
39  writeBits=0FH;
40  stopBits=0F0H;
41  bufSizeBits=0FH;
42  buf512=0;
43  buf1024=1;
44  buf2048=2;
45  buf4096=3;
46  buf8000=4;
47  buf16000=5;
48
49 TYPE
50  Remember=RECORD
51   nextRemember:RememberPtr;
52   rememberSize:LONGCARD;
53   memory:ADDRESS;
54  END;
55
56 CONST
57  deadendAlert=80000000H; recoveryAlert=0;
```

```
1
2 TYPE
3  DisplayMode=(hiresPick,lowresPick);
4
5 CONST
6  dModeCount=ORD(MAX(DisplayMode))+1;
7  eventMax=10;
8
9 TYPE
10  Res=(hiresGadget,lowresGadget);
11
12 CONST
13  resCount=ORD(MAX(Res))+1;
14
15 TYPE
16  Gadgets=(
17   upFrontGadget,downBackGadget,sizeGadget,closeGadget,
18   dragGadget,sUpFrontGadget,sDownBackGadget,sDragGadget
19  );
20
21 CONST
22  gadgetCount=ORD(MAX(Gadgets))+1;
23
24 TYPE
25  ILocks=(
26   iStateLock,layerInfoLock,gadgetsLock,layerRomLock,viewLock,
27   iBaseLock,rpLock
28  );
29 CONST
30  numILocks=ORD(MAX(ILocks))+1;
31
32 TYPE
33  FatIntuiMessage=RECORD
34   intuiMessage:IntuiMessage;
35   prevKeys:LONGCARD;
36  END;
37  IBox=RECORD
38   left:INTEGER;
39   top:INTEGER;
40   width:INTEGER;
41   height:INTEGER;
42  END;
43  Point=RECORD
44   x:INTEGER;
45   y:INTEGER;
46  END;
47  PenPair=RECORD
48   detailPen:UByte;
49   blockPen:UByte;
50  END;
51  GListEnv=RECORD
52   screen:ScreenPtr;
53   window:WindowPtr;
54   requester:RequesterPtr;
55   rastPort:RastPortPtr;
56   layer:LayerPtr;
57   gzzLayer:LayerPtr;
```

```
 1   pens:PenPair;
 2   domain:IBox;
 3   gzzDims:IBox;
 4  END;
 5  GListEnvPtr=POINTER TO GListEnv;
 6  GadgetInfo=RECORD
 7   environ:GListEnvPtr;
 8   gadget:GadgetPtr;
 9   box:IBox;
10   container:IBox;
11   layer:LayerPtr;
12   newKnob:IBox;
13  END;
14
15 CONST
16  numIEvents=4;
17
18 TYPE
19  IntuitionBase=RECORD
20   libNode:Library;
21   viewLord:View;
22   activeWindow:WindowPtr;
23   activeScreen:ScreenPtr;
24   firstScreen:ScreenPtr;
25   flags:LONGSET;
26   mouseY:INTEGER;
27   mouseX:INTEGER;
28   seconds:LONGCARD;
29   micros:LONGCARD;
30   minXMouse:INTEGER;
31   maxXMouse:INTEGER;
32   minYMouse:INTEGER;
33   maxYMouse:INTEGER;
34   startSecs:LONGCARD;
35   startMicros:LONGCARD;
36   sysBase:ADDRESS;
37   gfxBase:GfxBasePtr;
38   layersBase:ADDRESS;
39   consoleDevice:ADDRESS;
40   aPointer:ADDRESS;
41   aPtrHeight:UByte;
42   aPtrWidth:[0..16];
43   aXOffset:UByte;
44   aYOffset:UByte;
45   menuDrawn:CARDINAL;
46   menuSelected:CARDINAL;
47   optionList:CARDINAL;
48   menuRPort:RastPort;
49   menuTmpRas:TmpRas;
50   itemCRect:ClipRect;
51   subCRect:ClipRect;
52   iBitMap:BitMap;
53   sBitMap:BitMap;
54   inputRequest:IOStdReq;
55   inputInterrupt:Interrupt;
56   eventKey:RememberPtr;
57   iEvents:ADDRESS;
```

```
1   eventCount:INTEGER;
2   ieBuffer:ARRAY [0..numIEvents-1] OF InputEvent;
3   activeGadget:GadgetPtr;
4   activePInfo:PropInfoPtr;
5   activeImage:ImagePtr;
6   gadgetEnv:GListEnv;
7   gadgetInfo:GadgetInfo;
8   knobOffset:Point;
9   getOKWindow:WindowPtr;
10  getOKMessage:IntuiMessagePtr;
11  setWExcept:CARDINAL;
12  gadgetReturn:CARDINAL;
13  stateReturn:CARDINAL;
14  rp:RastPortPtr;
15  iTmpRas:TmpRas;
16  oldClipRegion:RegionPtr;
17  oldScroll:Point;
18  iFrame:IBox;
19  hthick:INTEGER;
20  vthick:INTEGER;
21  frameChange:PROC;
22  sizeDrag:PROC;
23  firstPt:Point;
24  oldPt:Point;
25  sysGadgets:ARRAY Res,Gadgets OF GadgetPtr;
26  checkImage:ARRAY Res OF ImagePtr;
27  amigaIcon:ARRAY Res OF ImagePtr;
28  aPattern:ARRAY [0..7] OF CARDINAL;
29  bPattern:ARRAY [0..3] OF CARDINAL;
30  iPointer:ADDRESS;
31  iPtrHeight:UByte;
32  iPtrWidth:[0..16];
33  iXOffset:UByte;
34  iYOffset:UByte;
35  doubleSeconds:LONGINT;
36  doubleMicros:LONGINT;
37  wBorLeft,
38  wBorTop,
39  wBorRight,
40  wBorBottom,
41  barVBorder,
42  barHBorder,
43  menuVBorder,
44  menuHBorder:ARRAY DisplayMode OF Byte;
45  color0:CARDINAL;
46  color1:CARDINAL;
47  color2:CARDINAL;
48  color3:CARDINAL;
49  color17:CARDINAL;
50  color18:CARDINAL;
51  color19:CARDINAL;
52  sysFont:TextAttr;
53  preferences:PreferencesPtr;
54  echoes:ADDRESS;
55  viewInitX:INTEGER;
56  viewInitY:INTEGER;
57  cursorDX:INTEGER;
```

```
  cursorDY:INTEGER;
  keyMap:KeyMapPtr;
  mouseYMinimum:INTEGER;
  errorX:INTEGER;
  errorY:INTEGER;
  ioExcess:TimeRequest;
  holdMinYMouse:INTEGER;
  wbPort:MsgPortPtr;
  fnkuhdPort:MsgPortPtr;
  wbMessage:IntuiMessage;
  hitScreen:ScreenPtr;
  simpleSprite:SimpleSpritePtr;
  attachedSSprite:SimpleSpritePtr;
  gotSpritel:BOOLEAN;
  semaphoreList:List;
  iSemaphore:ARRAY ILocks OF SignalSemaphore;
  maxDisplayHeight:INTEGER;
  maxDisplayRow:INTEGER;
  reserved:ARRAY [0..7] OF LONGCARD;
 END;
 IntuitionBasePtr=POINTER TO IntuitionBase;

(*
 * Die Prozedure AllocRemember, OpenScreen und OpenWindow
 * haben eine VAR Parameter.
 *)

PROCEDURE ActivateGadget(
            gadget{8}:GadgetPtr;
            window{9}:WindowPtr;
            requester{10}:RequesterPtr):BOOLEAN; CODE -462;
PROCEDURE ActivateWindow(window{8}:WindowPtr); CODE -450;
PROCEDURE AddGadget(window{8}:WindowPtr;
                    gadget{9}:GadgetPtr;
                    position{0}:INTEGER):INTEGER; CODE -42;
PROCEDURE AddGList(
            window{8}:WindowPtr;
            gadget{9}:GadgetPtr;
            position{0}:INTEGER;
            numGad{1}:INTEGER;
            requester{10}:RequesterPtr):INTEGER; CODE -438;
PROCEDURE AllocRemember(
            VAR rememberKey{8}:ADDRESS;
            size{0}:LONGCARD;
            flags{1}:MemReqSet):ADDRESS; CODE -396;
PROCEDURE AlohaWorkbench(wbPort{8}:MsgPortPtr); CODE -402;
PROCEDURE AutoRequest(window{8}:WindowPtr;
                      bodyText{9}:IntuiTextPtr;
                      positiveText{10}:IntuiTextPtr;
                      negativeText{11}:IntuiTextPtr;
                      positiveFlags{0}:IDCMPFlagSet;
                      negativeFlags{1}:IDCMPFlagSet;
                      width{2}:INTEGER;
                      height{3}:INTEGER):BOOLEAN; CODE -348;
PROCEDURE BeginRefresh(window{8}:WindowPtr); CODE -354;
PROCEDURE BuildSysRequest(
            window{8}:WindowPtr;
```

```
1              bodyText{9}:IntuiTextPtr;
2              positiveText{10}:IntuiTextPtr;
3              negativeText{11}:IntuiTextPtr;
4              idcmpFlags{0}:IDCMPFlagSet;
5              width{1}:INTEGER;
6              height{2}:INTEGER):WindowPtr; CODE -360;
7  PROCEDURE ClearDMRequest(
8              window{8}:WindowPtr):BOOLEAN; CODE -48;
9  PROCEDURE ClearMenuStrip(window{8}:WindowPtr); CODE -54;
10 PROCEDURE ClearPointer(window{8}:WindowPtr); CODE -60;
11 PROCEDURE CloseScreen(screen{8}:ScreenPtr); CODE -66;
12 PROCEDURE CloseWindow(window{8}:WindowPtr); CODE -72;
13 PROCEDURE CloseWorkBench():BOOLEAN; CODE -78;
14 PROCEDURE CurrentTime(seconds{8}:ADDRESS;
15                       micros{9}:ADDRESS); CODE -84;
16 PROCEDURE DisplayAlert(alertNumber{0}:LONGCARD;
17                        string{8}:ADDRESS;
18                        height{1}:LONGCARD):BOOLEAN; CODE -90;
19 PROCEDURE DisplayBeep(screen{8}:ScreenPtr); CODE -96;
20 PROCEDURE DoubleClick(
21             startSecs{0}:LONGINT;
22             startMicros{1}:LONGINT;
23             currentSecs{2}:LONGINT;
24             currentMicros{3}:LONGINT):BOOLEAN; CODE -102;
25 PROCEDURE DrawBorder(rastPort{8}:RastPortPtr;
26                      border{9}:BorderPtr;
27                      leftOffset{0}:INTEGER;
28                      topOffset{1}:INTEGER); CODE -108;
29 PROCEDURE DrawImage(rastPort{8}:RastPortPtr;
30                     image{9}:ImagePtr;
31                     leftOffset{0}:INTEGER;
32                     topOffset{1}:INTEGER); CODE -114;
33 PROCEDURE EndRefresh(window{8}:WindowPtr;
34                       complete{0}:BOOLEAN); CODE -366;
35 PROCEDURE EndRequest(requester{8}:RequesterPtr;
36                       window{9}:WindowPtr); CODE -120;
37 PROCEDURE FreeRemember(rememberKey{8}:ADDRESS;
38                         reallyForget{0}:BOOLEAN); CODE -408;
39 PROCEDURE FreeSysRequest(window{8}:WindowPtr); CODE -372;
40 PROCEDURE GetDefPrefs(prefBuffer{8}:PreferencesPtr;
41                        size{0}:INTEGER); CODE -126;
42 PROCEDURE GetPrefs(prefBuffer{8}:PreferencesPtr;
43                    size{0}:INTEGER); CODE -132;
44 PROCEDURE GetScreenData(
45             buffer{8}:ADDRESS;
46             size{0}:CARDINAL;
47             type{1}:ScreenFlagSet;
48             screen{9}:ScreenPtr):BOOLEAN; CODE -426;
49 PROCEDURE InitRequester(requester{8}:RequesterPtr); CODE -138;
50 PROCEDURE IntuiTextLength(
51             iText{8}:IntuiTextPtr):INTEGER; CODE -330;
52 PROCEDURE Intuition(inputEvent{8}:InputEventPtr); CODE -36;
53 PROCEDURE ItemAddress(
54             menuStrip{8}:MenuPtr;
55             menuNumber{0}:CARDINAL):MenuItemPtr; CODE -144;
56 PROCEDURE LockIBase(
57             lockNumber{0}:LONGCARD):LONGCARD; CODE -414;
```

```
1 PROCEDURE MakeScreen(screen{8}:ScreenPtr); CODE -378;
2 PROCEDURE ModifyIDCMP(window{8}:WindowPtr;
3                       idcmpFlags{0}:IDCMPFlagSet); CODE -150;
4 PROCEDURE ModifyProp(gadget{8}:GadgetPtr;
5                      window{9}:WindowPtr;
6                      requester{10}:RequesterPtr;
7                      flags{0}:PropInfoFlagSet;
8                      horizPot{1}:CARDINAL;
9                      vertPot{2}:CARDINAL;
10                     horizBody{3}:CARDINAL;
11                     vertBody{4}:CARDINAL); CODE -156;
12 PROCEDURE MoveScreen(screen{8}:ScreenPtr;
13                      deltaX{0}:INTEGER;
14                      deltaY{1}:INTEGER); CODE -162;
15 PROCEDURE MoveWindow(window{8}:WindowPtr;
16                      deltaX{0}:INTEGER;
17                      deltaY{1}:INTEGER); CODE -168;
18 PROCEDURE NewModifyProp(gadget{8}:GadgetPtr;
19                         window{9}:WindowPtr;
20                         requester{10}:RequesterPtr;
21                         flags{0}:PropInfoFlagSet;
22                         horizPot{1}:CARDINAL;
23                         vertPot{2}:CARDINAL;
24                         horizBody{3}:CARDINAL;
25                         vertBody{4}:CARDINAL;
26                         numGad{5}:INTEGER); CODE -468;
27 PROCEDURE OffGadget(gadget{8}:GadgetPtr;
28                     window{9}:WindowPtr;
29                     requester{10}:RequesterPtr); CODE -174;
30 PROCEDURE OffMenu(window{8}:WindowPtr;
31                    menuNumber{0}:CARDINAL); CODE -180;
32 PROCEDURE OnGadget(gadget{8}:GadgetPtr;
33                     window{9}:WindowPtr;
34                     requester{10}:RequesterPtr); CODE -186;
35 PROCEDURE OnMenu(window{8}:WindowPtr;
36                   menuNumber{0}:CARDINAL); CODE -192;
37 PROCEDURE OpenIntuition():IntuitionBasePtr; CODE -30;
38 PROCEDURE OpenScreen(
39             VAR newScreen{8}:NewScreen):ScreenPtr; CODE -198;
40 PROCEDURE OpenWindow(
41             VAR newWindow{8}:NewWindow):WindowPtr; CODE -204;
42 PROCEDURE OpenWorkBench():ScreenPtr; CODE -210;
43 PROCEDURE PrintIText(rastPort{8}:RastPortPtr;
44                       iText{9}:IntuiTextPtr;
45                       leftOffset{0}:INTEGER;
46                       topOffset{1}:INTEGER); CODE -216;
47 PROCEDURE RefreshGadgets(
48             gadgets{8}:GadgetPtr;
49             window{9}:WindowPtr;
50             requester{10}:RequesterPtr); CODE -222;
51 PROCEDURE RefreshGList(gadgets{8}:GadgetPtr;
52                        window{9}:WindowPtr;
53                        requester{10}:RequesterPtr;
54                        numGad{0}:INTEGER); CODE -432;
55 PROCEDURE RefreshWindowFrame(window{8}:WindowPtr); CODE -456;
56 PROCEDURE RemakeDisplay(); CODE -384;
57 PROCEDURE RemoveGadget(
```

```
            window{8}:WindowPtr;
            gadget{9}:GadgetPtr):INTEGER; CODE -228;
PROCEDURE RemoveGList(window{8}:WindowPtr;
                      gadget{9}:GadgetPtr;
                      numgad{0}:INTEGER):INTEGER; CODE -444;
PROCEDURE ReportMouse(window{8}:WindowPtr;
                      boolean{0}:BOOLEAN); CODE -234;
PROCEDURE Request(requester{8}:RequesterPtr;
                  window{9}:WindowPtr):BOOLEAN; CODE -240;
PROCEDURE RethinkDisplay(); CODE -390;
PROCEDURE ScreenToBack(screen{8}:ScreenPtr); CODE -246;
PROCEDURE ScreenToFront(screen{8}:ScreenPtr); CODE -252;
PROCEDURE SetDMRequest(
            window{8}:WindowPtr;
            dmRequester{9}:RequesterPtr):BOOLEAN; CODE -258;
PROCEDURE SetMenuStrip(window{8}:WindowPtr;
                       menu{9}:MenuPtr):BOOLEAN; CODE -264;
PROCEDURE SetPointer(window{8}:WindowPtr;
                     pointer{9}:ADDRESS;
                     height{0}:INTEGER;
                     width{1}:INTEGER;
                     xOffset{2}:INTEGER;
                     yOffset{3}:INTEGER); CODE -270;
PROCEDURE SetPrefs(prefBuffer{8}:PreferencesPtr;
                   Size{0}:INTEGER;
                   Inform{1}:BOOLEAN); CODE -324;
PROCEDURE SetWindowTitles(
            window{8}:WindowPtr;
            windowTitle{9}:ADDRESS;
            screenTitle{10}:ADDRESS); CODE -276;
PROCEDURE ShowTitle(screen{8}:ScreenPtr;
                    showIt{0}:BOOLEAN); CODE -282;
PROCEDURE SizeWindow(window{8}:WindowPtr;
                     deltaX{0}:INTEGER;
                     deltaY{1}:INTEGER); CODE -288;
PROCEDURE UnlockIBase(lock{8}:LONGCARD); CODE -420;
PROCEDURE ViewAddress():ViewPtr; CODE -294;
PROCEDURE ViewPortAddress(
            window{8}:WindowPtr):ViewPortPtr; CODE -300;
PROCEDURE WBenchToBack():BOOLEAN; CODE -336;
PROCEDURE WBenchToFront():BOOLEAN; CODE -342;
(*
 * Den Paramentern maxWidth und maxHeight darf man auch den
 * Wert -1 zuweisen, falls man die Window Grösse nicht
 * beschränken will. Damit dies nicht zu einem Laufzeitfehler
 * führt wurden diese Parameter im Gegensatz zu den
 * C Deklarationen als INTEGER und nicht als CARDINAL
 * deklariert.
 *)
PROCEDURE WindowLimits(
            window{8}:WindowPtr;
            minWidth{0}:INTEGER;
            minHeight{1}:INTEGER;
            maxWidth{2}:INTEGER;
            maxHeight{3}:INTEGER):BOOLEAN; CODE -318;
PROCEDURE WindowToBack(window{8}:WindowPtr); CODE -306;
PROCEDURE WindowToFront(window{8}:WindowPtr); CODE -312;
```

```
1
2 END Intuition.
3
```

Keyboard

```
 5 (* $M- *)
 6 DEFINITION MODULE Keyboard;
 7
 8 FROM Exec IMPORT
 9  nonstd;
10
11 CONST
12  keyboardName="keyboard.device";
13  readEvent=nonstd+0;
14  readMatrix=nonstd+1;
15  addResetHandler=nonstd+2;
16  remResetHandler=nonstd+3;
17  resetHandlerDone=nonstd+4;
18
19 END Keyboard.
20
```

KeyMap

```
(* $M- *)
DEFINITION MODULE KeyMap;

FROM SYSTEM IMPORT
 ADDRESS,LONGSET;
FROM Exec IMPORT
 Node,List;

TYPE
 KeyMapTypes=(shift,alt,control,downup,kmp4,dead,string,nop);
 KeyMapTypeSet=SET OF KeyMapTypes;
 DeadPrefixBytes=(dpbMod,dpb1,dpb2,dpbDead);
 DeadPrefixByteSet=SET OF DeadPrefixBytes;

CONST
 dp2dIndexMask=0FH;
 dp2dFacShift=4;
 maxKeys=64;
 noQual=KeyMapTypeSet{};
 vanilla=KeyMapTypeSet{shift,alt,control};

TYPE
 BitTable=
  ARRAY [0..maxKeys DIV (8*SIZE(LONGSET))-1] OF LONGSET;
 BitTablePtr=POINTER TO BitTable;
 KeyInfo=RECORD
  CASE :INTEGER OF
  | 0: ch:ARRAY [0..3] OF CHAR;
  | 1: st:ADDRESS
  END
 END;
 Types=ARRAY [0..maxKeys-1] OF KeyMapTypeSet;
 TypesPtr=POINTER TO Types;
 Info=ARRAY [0..maxKeys-1] OF KeyInfo;
 InfoPtr=POINTER TO Info;
 KeyMap=RECORD
  loKeyMapTypes:TypesPtr;
  loKeyMap:InfoPtr;
  loCapsable:BitTablePtr;
  loRepeatable:BitTablePtr;
  hiKeyMapTypes:TypesPtr;
  hiKeyMap:InfoPtr;
  hiCapsable:BitTablePtr;
  hiRepeatable:BitTablePtr;
 END;
 KeyMapPtr=POINTER TO KeyMap;
 KeyMapNode=RECORD
  node:Node;
  keyMap:KeyMap;
 END;
 KeyMapResource=RECORD
  node:Node;
  list:List;
```

```
1  END;
2
3 END KeyMap.
4
```

Layers

```
 5 DEFINITION MODULE Layers {"layers.library", 33};
 6
 7 FROM SYSTEM IMPORT
 8  LONGSET;
 9 FROM Graphics IMPORT
10  ClipRectPtr,BitMapPtr,LayerInfoPtr,LayerPtr,RastPortPtr,
11  RegionPtr;
12
13 PROCEDURE BeginUpdate(l{8}:LayerPtr):LONGINT; CODE -78;
14 PROCEDURE BehindLayer(l{9}:LayerPtr):LONGINT; CODE -54;
15 PROCEDURE CreateBehindLayer(
16             li{8}:LayerInfoPtr;
17             bm{9}:BitMapPtr;
18             x0{0}:LONGINT;
19             y0{1}:LONGINT;
20             x1{2}:LONGINT;
21             y1{3}:LONGINT;
22             flags{4}:LONGSET;
23             bm2{10}:BitMapPtr):LayerPtr; CODE -42;
24 PROCEDURE CreateUpfrontLayer(
25             li{8}:LayerInfoPtr;
26             bm{9}:BitMapPtr;
27             x0{0}:LONGINT;
28             y0{1}:LONGINT;
29             x1{2}:LONGINT;
30             y1{3}:LONGINT;
31             flags{4}:LONGSET;
32             bm2{10}:BitMapPtr):LayerPtr; CODE -36;
33 PROCEDURE DeleteLayer(l{9}:LayerPtr):LONGINT; CODE -90;
34 PROCEDURE DisposeLayerInfo(li{8}:LayerInfoPtr); CODE -150;
35 PROCEDURE EndUpdate(l{8}:LayerPtr; flag{0}:BOOLEAN); CODE -84;
36 (*VERALTET*)PROCEDURE FattenLayerInfo(
37                          li{8}:LayerInfoPtr); CODE -156;
38 (*VERALTET*)PROCEDURE InitLayers(
39               li{8}:LayerInfoPtr); CODE -30;
40 PROCEDURE InstallClipRegion(
41             l{8}:LayerPtr;
42             region{9}:RegionPtr):RegionPtr; CODE -174;
43 PROCEDURE LockLayer(l{9}:LayerPtr); CODE -96;
44 PROCEDURE LockLayerInfo(li{8}:LayerInfoPtr); CODE -120;
45 PROCEDURE LockLayers(li{8}:LayerInfoPtr); CODE -108;
46 PROCEDURE MoveLayer(l{9}:LayerPtr;
47                      dx{0},dy{1}:LONGINT):LONGINT; CODE -60;
48 PROCEDURE MoveLayerInFrontOf(
49             l{8},target{9}:LayerPtr):LONGINT; CODE -168;
50 PROCEDURE NewLayerInfo():LayerInfoPtr; CODE -144;
51 PROCEDURE ScrollLayer(
52             l{9}:LayerPtr; dx{0},dy{1}:LONGINT); CODE -72;
53 PROCEDURE SizeLayer(l{9}:LayerPtr;
54                      dx{0},dy{1}:LONGINT):LONGINT; CODE -66;
55 PROCEDURE SwapBitsRastPortClipRect(
56             rp{8}:RastPortPtr;
57             cr{9}:ClipRectPtr); CODE -126;
```

```
1  (*VERALTET*)PROCEDURE ThinLayerInfo(
2                          li{8}:LayerInfoPtr); CODE -162;
3  PROCEDURE UnlockLayer(l{8}:LayerPtr); CODE -102;
4  PROCEDURE UnlockLayerInfo(li{8}:LayerInfoPtr); CODE -138;
5  PROCEDURE UnlockLayers(li{8}:LayerInfoPtr); CODE -114;
6  PROCEDURE UpfrontLayer(l{9}:LayerPtr):LONGINT; CODE -48;
7  PROCEDURE WhichLayer(li{8}:LayerInfoPtr;
8                       x{0},y{1}:LONGINT):LayerPtr; CODE -132;
9
10 END Layers.
11
```

MathIEEEDoubTrans

```
 5 DEFINITION MODULE
 6  MathIEEEDoubTrans {"mathieeedoubtrans.library",34};
 7
 8 PROCEDURE Acos(x{0}:LONGREAL):LONGREAL; CODE -120;
 9 PROCEDURE Asin(x{0}:LONGREAL):LONGREAL; CODE -114;
10 PROCEDURE Atan(x{0}:LONGREAL):LONGREAL; CODE -30;
11 PROCEDURE Cos(x{0}:LONGREAL):LONGREAL; CODE -42;
12 PROCEDURE Cosh(x{0}:LONGREAL):LONGREAL; CODE -66;
13 PROCEDURE Exp(x{0}:LONGREAL):LONGREAL; CODE -78;
14 PROCEDURE Fieee(x{0}:REAL):LONGREAL; CODE -108;
15 PROCEDURE Log(x{0}:LONGREAL):LONGREAL; CODE -84;
16 PROCEDURE Log10(x{0}:LONGREAL):LONGREAL; CODE -126;
17 PROCEDURE Pow(exp{0},x{2}:LONGREAL):LONGREAL; CODE -90;
18 PROCEDURE Sin(x{0}:LONGREAL):LONGREAL; CODE -36;
19 PROCEDURE Sincos(x{0}:LONGREAL;
20                  VAR cos{8}:LONGREAL):LONGREAL; CODE -54;
21 PROCEDURE Sinh(x{0}:LONGREAL):LONGREAL; CODE -60;
22 PROCEDURE Sqrt(x{0}:LONGREAL):LONGREAL; CODE -96;
23 PROCEDURE Tan(x{0}:LONGREAL):LONGREAL; CODE -48;
24 PROCEDURE Tanh(x{0}:LONGREAL):LONGREAL; CODE -72;
25 PROCEDURE Tieee(x{0}:LONGREAL):REAL; CODE -102;
26
27 END MathIEEEDoubTrans.
28
```

MathTrans

```
 5 DEFINITION MODULE MathTrans {"mathtrans.library",33};
 6
 7 FROM SYSTEM IMPORT FFP;
 8
 9 PROCEDURE Acos(x{0}:FFP):FFP; CODE -120;
10 PROCEDURE Asin(x{0}:FFP):FFP; CODE -114;
11 PROCEDURE Atan(x{0}:FFP):FFP; CODE -30;
12 PROCEDURE Cos(x{0}:FFP):FFP; CODE -42;
13 PROCEDURE Cosh(x{0}:FFP):FFP; CODE -66;
14 PROCEDURE Exp(x{0}:FFP):FFP; CODE -78;
15 PROCEDURE Fieee(x{0}:REAL):FFP; CODE -108;
16 PROCEDURE Log(x{0}:FFP):FFP; CODE -84;
17 PROCEDURE Log10(x{0}:FFP):FFP; CODE -126;
18 PROCEDURE Pow(exp{0},x{1}:FFP):FFP; CODE -90;
19 PROCEDURE Sin(x{0}:FFP):FFP; CODE -36;
20 PROCEDURE Sincos(x{0}:FFP; VAR cos{1}:FFP):FFP; CODE -54;
21 PROCEDURE Sinh(x{0}:FFP):FFP; CODE -60;
22 PROCEDURE Sqrt(x{0}:FFP):FFP; CODE -96;
23 PROCEDURE Tan(x{0}:FFP):FFP; CODE -48;
24 PROCEDURE Tanh(x{0}:FFP):FFP; CODE -72;
25 PROCEDURE Tieee(x{0}:FFP):REAL; CODE -102;
26
27 END MathTrans.
28
```

Narrator

```
 5 (* $M- *)
 6 DEFINITION MODULE Narrator;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS,BYTE;
10 FROM Exec IMPORT
11  IOStdReq,UByte;
12
13 CONST
14  narratorName="narrator.device";
15  noMem=-2;
16  noAudLib=-3;
17  makeBad=-4;
18  unitErr=-5;
19  cantAlloc=-6;
20  unimpl=-7;
21  noWrite=-8;
22  expunged=-9;
23  phonErr=-20;
24  rateErr=-21;
25  pitchErr=-22;
26  sexErr=-23;
27  modeErr=-24;
28  freqErr=-25;
29  volErr=-26;
30
31 CONST
32  male=0;
33  female=1;
34  natural=0;
35  robotic=1;
36  defPitch=110;
37  defRate=150;
38  defVol=64;
39  defFreq=22200;
40  defSex=male;
41  defMode=natural;
42  minRate=40;
43  maxRate=400;
44  minPitch=65;
45  maxPitch=320;
46  minFreq=5000;
47  maxFreq=28000;
48  minVol=0;
49  maxVol=64;
50
51 TYPE
52  IONarrator=RECORD
53   message:IOStdReq;
54   rate:CARDINAL;
55   pitch:CARDINAL;
56   mode:CARDINAL;
57   sex:CARDINAL;
```

```
1    chMasks:ADDRESS;
2    nmMasks:CARDINAL;
3    volume:CARDINAL;
4    sampFreq:CARDINAL;
5    mouths:UByte;
6    chanMask:BYTE;
7    numChan:BYTE;
8    pad:BYTE;
9   END;
10  IONarratorPtr=POINTER TO IONarrator;
11  Mouth=RECORD
12   voice:Narrator;
13   width:UByte;
14   height:UByte;
15   shape:BYTE;
16   pad:BYTE;
17  END;
18  MouthPtr=POINTER TO Mouth;
19
20 END Narrator.
21
```

Parallel

```
(*$M-*)
DEFINITION MODULE Parallel;

FROM Exec IMPORT
 IOStdReq,nonstd;

CONST
 parallelName="parallel.device";
 query=nonstd;
 setParams=nonstd+1;

TYPE
 IOPArray=RECORD
  pTermArray0:LONGCARD;
  pTermArray1:LONGCARD;
 END;
 ParErr=(
  pe0,devBusy,bufTooBig,invParam,lineErr,notOpen,portReset,
  initErr
 );
 ParFlags=(pf0,eofMode,pf2,radBoogie,pf4,shared);
 ParFlagSet=SET OF ParFlags;
 Status=(pSel,paperOut,pBusy,rwDir,active,abort,queued);
 StatusSet=SET OF Status;
 IOParallel=RECORD
  ioPar:IOStdReq;
  pExtFlags:LONGCARD;
  status:StatusSet;
  parFlags:ParFlagSet;
  pTermArray:IOPArray;
 END;
 IOParallelPtr=POINTER TO IOParallelPar;

END Parallel.

```

Printer

```
 5 (*$M-*)
 6 DEFINITION MODULE Printer;
 7
 8 FROM SYSTEM IMPORT
 9  CAST;
10 FROM Exec IMPORT
11  nonstd,DevicePtr,IOFlagSet,Message,UnitPtr,UByte;
12 FROM Graphics IMPORT
13  ColorMapPtr,RastPortPtr,ViewModeSet;
14
15 CONST
16  printerName="printer.device";
17  rawWrite=nonstd+0;
18  prtCommand=nonstd+1;
19  dumpRPort=nonstd+2;
20
21  ris=0;      (* ESCc  reset                      ISO *)
22  rin=1;      (* ESC#1 initialize                 +++ *)
23  ind=2;      (* ESCD  lf                         ISO *)
24  nel=3;      (* ESCE  return,lf                  ISO *)
25  ri=4;       (* ESCM  reverse lf                 ISO *)
26
27  sgr0=5;     (* ESC[0m normal char set           ISO *)
28  sgr3=0;     (* ESC[3m italics on                ISO *)
29  sgr23=7;    (* ESC[23m italics off              ISO *)
30  sgr4=8;     (* ESC[4m underline=0;              ISO *)
31  sgr24=9;    (* ESC[24m underline off            ISO *)
32  sgr1=10;    (* ESC[1m boldface on               ISO *)
33  sgr22=11;   (* ESC[22m boldface off             ISO *)
34  sfc=12;     (* SGR30-39  set foreground color ISO *)
35  sbc=13;     (* SGR40-49  set background color ISO *)
36
37  shorp0=14; (* ESC[0w normal pitch              DEC *)
38  shorp2=15; (* ESC[2w elite on                  DEC *)
39  shorp1=16; (* ESC[1w elite off                 DEC *)
40  shorp4=17; (* ESC[4w condensed fine on         DEC *)
41  shorp3=18; (* ESC[3w condensed off             DEC *)
42  shorp6=19; (* ESC[6w enlarged on               DEC *)
43  shorp5=20; (* ESC[5w enlarged off              DEC *)
44
45  den6=  21; (* ESC[6"z shadow print on          DEC (sort of) *)
46  den5=  22; (* ESC[5"z shadow print off         DEC *)
47  den4=  23; (* ESC[4"z doublestrike on          DEC *)
48  den3=  24; (* ESC[3"z doublestrike off         DEC *)
49  den2=  25; (* ESC[2"z  NLQ on                  DEC *)
50  den1=  26; (* ESC[1"z  NLQ off                 DEC *)
51
52  sus2=  27; (* ESC[2v superscript on            +++ *)
53  sus1=  28; (* ESC[1v superscript off           +++ *)
54  sus4=  29; (* ESC[4v subscript on              +++ *)
55  sus3=  30; (* ESC[3v subscript off             +++ *)
56  sus0=  31; (* ESC[0v normalize the line        +++ *)
57  plu=   32; (* ESCL  partial line up            ISO *)
```

```
 1  pld=    33; (* ESCK  partial line down          ISO *)
 2
 3  fnt0=   34; (* ESC(B US char set                DEC *)
 4  fnt1=   35; (* ESC(R French char set            DEC *)
 5  fnt2=   36; (* ESC(K German char set            DEC *)
 6  fnt3=   37; (* ESC(A UK char set                DEC *)
 7  fnt4=   38; (* ESC(E Danish I char set          DEC *)
 8  fnt5=   39; (* ESC(H Sweden char set            DEC *)
 9  fnt6=   40; (* ESC(Y Italian char set           DEC *)
10  fnt7=   41; (* ESC(Z Spanish char set           DEC *)
11  fnt8=   42; (* ESC(J Japanese char set          +++ *)
12  fnt9=   43; (* ESC(6 Norweign char set          DEC *)
13  fnt10=  44; (* ESC(C Danish II char set         +++ *)
14
15  prop2=  45; (* ESC[2p  proportional on          +++ *)
16  prop1=  46; (* ESC[1p  proportional off         +++ *)
17  prop0=  47; (* ESC[0p  proportional clear       +++ *)
18  tss=    48; (* ESC[n E set proportional offset ISO *)
19  jfy5=   49; (* ESC[5 F auto left justify        ISO *)
20  jfy7=   50; (* ESC[7 F auto right justify       ISO *)
21  jfy6=   51; (* ESC[6 F auto full justify        ISO *)
22  jfy0=   52; (* ESC[0 F auto justify off         ISO *)
23  jfy3=   53; (* ESC[3 F letter space (justify) ISO (special) *)
24  jfy1=   54; (* ESC[1 F word fill(auto center) ISO (special) *)
25
26  verp0=  55; (* ESC[0z  1/8" line spacing        +++ *)
27  verp1=  56; (* ESC[1z  1/6" line spacing        +++ *)
28  slpp=   57; (* ESC[nt  set form length n        DEC *)
29  perf=   58; (* ESC[nq  perf skip n (n>0)        +++ *)
30  perf0=  59; (* ESC[0q  perf skip off            +++ *)
31
32  lms=    60; (* ESC#9   Left margin set          +++ *)
33  rms=    61; (* ESC#0   Right margin set         +++ *)
34  tms=    62; (* ESC#8   Top margin set           +++ *)
35  bms=    63; (* ESC#2   Bottom marg set          +++ *)
36  stbm=   64; (* ESC[Pn1;Pn2r  T&B margins        DEC *)
37  slrm=   65; (* ESC[Pn1;Pn2s  L&R margin         DEC *)
38  cam=    66; (* ESC#3   Clear margins            +++ *)
39
40  hts=    67; (* ESCH    Set horiz tab           ISO *)
41  vts=    68; (* ESCJ    Set vertical tabs       ISO *)
42  tbc0=   69; (* ESC[0g  Clr horiz tab           ISO *)
43  tbc3=   70; (* ESC[3g  Clear all h tab         ISO *)
44  tbc1=   71; (* ESC[1g  Clr vertical tabs       ISO *)
45  tbc4=   72; (* ESC[4g  Clr all v tabs          ISO *)
46  tbcall=73; (* ESC#4   Clr all h & v tabs      +++ *)
47  tbsall=74; (* ESC#5   Set default tabs        +++ *)
48  extend=75; (* ESC[Pn"x extended commands      +++ *)
49
50 TYPE
51  Error=(
52   e0,cancel,notGraphics,invertHam,badDimension,dimensionOvflow,
53   internalMemory,bufferMemory
54  );
55  IOPrinter=RECORD
56   message:Message;
57   device:DevicePtr;
```

```
1    unit:UnitPtr;
2    command:CARDINAL;
3    flags:IOFlagSet;
4    error:Error;
5    prtCommand:CARDINAL;
6    parm0:UByte;
7    parm1:UByte;
8    parm2:UByte;
9    parm3:UByte;
10  END;
11  IOPrinterPtr=POINTER TO IOPrtCmdReq;
12  Special=(
13   milCols,milRows,fullCols,fullRows,fracCols,fracRows,center,
14   aspect,density1,density2,density4
15  );
16  SpecialSet=SET OF Special;
17
18 CONST
19  density3=SpecialSet{density1,density2};
20  densityMask=CAST(SpecialSet,0F00H);
21
22 TYPE
23  IODRPReq=RECORD
24   message:Message;
25   device:DevicePtr;
26   unit:UnitPtr;
27   command:CARDINAL;
28   flags:IOFlagSet;
29   error:Error;
30   rastPort:RastPortPtr;
31   colorMap:ColorMapPtr;
32   modesHi:CARDINAL;
33   modes:ViewModeSet;
34   srcX:CARDINAL;
35   srcY:CARDINAL;
36   srcWidth:CARDINAL;
37   srcHeight:CARDINAL;
38   destCols:LONGINT;
39   destRows:LONGINT;
40   special:SpecialSet;
41  END;
42  IODRPReqPtr=POINTER TO IODRPReq;
43
44 END Printer.
45
```

PrtBase

```
 5 (*$M-*)
 6 DEFINITION MODULE PrtBase;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS,BYTE;
10 FROM Exec IMPORT
11  Library,MsgPort,Task,UByte;
12 FROM Intuition IMPORT
13  Preferences;
14 FROM Parallel IMPORT
15  IOExtPar;
16 FROM Serial IMPORT
17  IOExtSer;
18 FROM Timer IMPORT
19  TimeRequest;
20
21 TYPE
22  DeviceData=RECORD
23   device:Library;
24   segment:ADDRESS;
25   execBase:ADDRESS;
26   cmdVectors:ADDRESS;
27   cmdBytes:ADDRESS;
28   numCommands:CARDINAL;
29  END;
30  DeviceDataPtr=POINTER TO DeviceData;
31
32 CONST
33  stkSize=0800H;
34
35 TYPE
36  PrinterSegmentPtr=POINTER TO PrinterSegment;
37  PrinterData=RECORD
38   device:DeviceData;
39   unit:MsgPort;
40   printerSegment:ADDRESS;
41   printerType:CARDINAL;
42   segmentData:PrinterSegmentPtr;
43   printBuf:ADDRESS;
44   pWrite:PROCEDURE():INTEGER;
45   pBothReady:PROCEDURE():INTEGER;
46   CASE :INTEGER OF
47   | 1: p0:IOExtPar;
48        p1:IOExtPar;
49   | 2: s0:IOExtSer;
50        s1:IOExtSer;
51   END;
52   tior:TimeRequest;
53   iorPort:MsgPort;
54   tc:Task;
55   stk:ARRAY [0..stkSize-1] OF BYTE;
56   flags:UByte;
57   pad:BYTE;
```

```
 1    preferences:Preferences;
 2    pWaitEnabled:UByte;
 3   END;
 4   PrinterDataPtr=POINTER TO PrinterData;
 5   PrinterClass=(gfx,color);
 6   PrinterClassSet=SET OF PrinterClass;
 7
 8  CONST
 9   bwAlpha=0;
10   bwGfx=1;
11   colorGfx=3;
12   bw=1;
13   ymc=2;
14   ymcBw=3;
15   ymcb=4;
16   fourColor=04H;
17   additive=8;
18   wb=9;
19   bgr=10;
20   bgrWb=11;
21   bgrw=12;
22
23  TYPE
24   PrinterExtendedData=RECORD
25    printerName:ADDRESS;
26    init:PROC;
27    expunge:PROC;
28    open:PROCEDURE():INTEGER;
29    close:PROC;
30    printerClass:BYTE; (* little bit strange *)
31    colorClass:UByte;
32    maxColumns:UByte;
33    numCharSets:UByte;
34    numRows:CARDINAL;
35    maxXDots:LONGCARD;
36    maxYDots:LONGCARD;
37    xDotsInch:CARDINAL;
38    yDotsInch:CARDINAL;
39    commands:ADDRESS;
40    doSpecial:PROCEDURE():INTEGER;
41    render:PROCEDURE():INTEGER;
42    timeoutSecs:LONGINT;
43    EightBitChars:ADDRESS;
44   END;
45   PrinterExtendedDataPtr=POINTER TO PrinterExtendedData;
46   PrinterSegment=RECORD
47    nextSegment:ADDRESS;
48    runAlert:LONGCARD;
49    version:CARDINAL;
50    revision:CARDINAL;
51    ped:PrinterExtendedData;
52   END;
53
54  END PrtBase.
55
```

Resources

```
(*$M-*)
DEFINITION MODULE Resources;

FROM SYSTEM IMPORT
 ADDRESS,LONGSET;
FROM Exec IMPORT
 base,vectSize,Interrupt,InterruptPtr,Library,LibraryPtr,
 List,Message,UByte;
FROM Hardware IMPORT
 CiaIcrFlags,CiaIcrFlagSet;

CONST
 ciaaName="ciaa.resource";
 ciabName="ciab.resource";

TYPE
 CiaResourcePtr=ADDRESS;

PROCEDURE AbleICR(
            cia{14}:CiaResourcePtr;
            mask{0}:CiaIcrFlagSet):CiaIcrFlagSet; CODE -18;
PROCEDURE AddICRVector(
            cia{14}:CiaResourcePtr; icrBit{0}:CiaIcrFlags;
            interrupt{9}:InterruptPtr):InterruptPtr; CODE -6;
PROCEDURE RemICRVector(cia{14}:CiaResourcePtr;
                       icrBit{0}:CiaIcrFlags;
                       interrupt{9}:InterruptPtr); CODE -12;
PROCEDURE SetICR(
            cia{14}:CiaResourcePtr;
            mask{0}:CiaIcrFlagSet):CiaIcrFlagSet; CODE -24;

TYPE
 DiscResourceUnit=RECORD
  message:Message;
  discBlock:Interrupt;
  discSync:Interrupt;
  index:Interrupt
 END;
 DiscResourceUnitPtr=POINTER TO DiscResourceUnit;
 DiscResourceFlags=(
  alloc0,alloc1,alloc2,alloc3,drf4,drf5,drf6,active
 );
 DiscResourceFlagSet=SET OF DiscResourceFlags;
 DiscResource=RECORD
  library:Library;
  current:DiscResourceUnitPtr;
  flags:DiscResourceFlagSet;
  pad:UByte;
  sysLib:LibraryPtr;
  ciaResource:LibraryPtr;
  unitID:ARRAY [alloc0..alloc3] OF LONGCARD;
  waiting:List;
  discBlock:Interrupt;
```

```
 1   discSync:Interrupt;
 2   index:Interrupt;
 3  END;
 4  DiscResourcePtr=POINTER TO DiscResource;
 5
 6 CONST
 7  diskName="disk.resource";
 8  dskDmaOff=4000H;
 9  amiga=0;
10  drt37422D2S=55555555H;
11  empty=0FFFFFFFFH;
12
13 PROCEDURE AllocUnit(disk{14}:DiscResourcePtr;
14                      unitNum{0}:LONGINT):LONGINT; CODE -6;
15 PROCEDURE FreeUnit(disk{14}:DiscResourcePtr;
16                     unitNum{0}:LONGINT); CODE -12;
17 PROCEDURE GetUnit(
18            disk{14}:DiscResourcePtr;
19            unitPointer{9}:DiscResourceUnitPtr
20           ):DiscResourceUnitPtr; CODE -18;
21 PROCEDURE GetUnitID(
22            disk{14}:DiscResourcePtr;
23            unitNum{0}:LONGINT):LONGCARD; CODE -30;
24 PROCEDURE GiveUnit(disk{14}:DiscResourcePtr); CODE -24;
25
26 CONST
27   miscName="misc.resource";
28
29 TYPE
30  ResourceTypes=(
31   serialPort,serialBits,parallelPort,parallelBits
32  );
33  MiscResource=RECORD
34   library: Library;
35   allocArray: ARRAY ResourceTypes OF ADDRESS;
36  END;
37  MiscResourcePtr=POINTER TO MiscResource;
38
39 PROCEDURE AllocMiscResource(
40            misc{14}:MiscResourcePtr;
41            unitNum{0}:LONGINT;
42            name{9}:ADDRESS):ADDRESS; CODE -6;
43 PROCEDURE FreeMiscResource(misc{14}:MiscResourcePtr;
44                            unitNum{0}:LONGINT); CODE -12;
45
46 CONST
47  potgoName="potgo.resource";
48
49 TYPE
50  PotgoBits=(
51   start,pb1,pb2,pb3,pb4,pb5,pb6,pb7,
52   datlx,outlx,datly,outly,datrx,outrx,datry,outry,pb16
53  );
54  PotgoBitSet=SET OF PotgoBits;
55
56 PROCEDURE AllocPotBits(
57            potgo{14}: ADDRESS;
```

```
1            bits{0}: PotgoBitSet): PotgoBitSet; CODE -6;
2 PROCEDURE FreePotBits(potgo{14}: ADDRESS;
3                        allocated{0}: PotgoBitSet); CODE -12;
4 PROCEDURE WritePotgo(potgo{14}: ADDRESS;
5                      word{0}: PotgoBitSet;
6                      mask{1}: PotgoBitSet); CODE -18;
7
8 END Resources.
9
```

Serial

```
 5 (*$M-*)
 6 DEFINITION MODULE Serial;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS,BYTE,LONGSET;
10 FROM Exec IMPORT
11  nonstd,IOFlagSet,IOStdReq,UByte;
12
13 CONST
14  serialName="serial.device";
15  query=nonstd;
16  break=nonstd+1;
17  setParams=nonstd+2;
18  mark=LONGSET{0};
19  mSpOn=LONGSET{1};
20  active=IOFlagSet{4};
21  abort=IOFlagSet{5};
22  queued=IOFlagSet{6};
23  bufrRead=IOFlagSet{7};
24
25 TYPE
26  IOTArray=RECORD
27   termArray0:LONGCARD;
28   termArray1:LONGCARD;
29  END;
30  SerFlags=(
31   partyOn,partyOdd,sevenWire,queuedBrk,radBoogie,shared,
32   eofMode,xDisabled
33  );
34  SerFlagSet=SET OF SerFlags;
35  Status=(
36   busy,paperOut,select,dataSetReady,clearToSend,carrierDetect,
37   readyToSend,dataTerminalReady,overrun,wroteBreak,readBreak,
38   xOffWrite,xOffRead
39  );
40  StatusSet=SET OF Status;
41  Error=(
42   e0,devBusy,baudMismatch,invBaud,bufErr,invParam,lineErr,
43   notOpen,portReset,parityErr,initErr,timerErr,bufOverflow,
44   nodsr,nocts,detectedBreak
45  );
46  IOSerial=RECORD
47   ioSer:IOStdReq;
48   ctlChar:LONGCARD;
49   rBufLen:LONGCARD;
50   extFlags:LONGSET;
51   baud:LONGCARD;
52   brkTime:LONGCARD;
53   termArray:IOTArray;
54   readLen:UByte;
55   writeLen:UByte;
56   stopBits:UByte;
57   serFlags:SerFlagSet;
```

```
1    status:StatusSet;
2   END;
3   IOSerialPtr=POINTER TO IOExtSer;
4
5 END Serial.
6
```

Timer

```
 5 (*$M-*)
 6 DEFINITION MODULE Timer;
 7
 8 FROM SYSTEM IMPORT
 9  ADDRESS;
10 FROM Exec IMPORT
11  nonstd,IORequest;
12
13 CONST
14  timerName="timer.device";
15  addRequest=nonstd+0;
16  getSysTime=nonstd+1;
17  setSysTime=nonstd+2;
18  microHz=0;
19  vBlank=1;
20
21 TYPE
22  TimeVal=RECORD
23   secs:LONGCARD;
24   micro:LONGCARD
25  END;
26  TimeValPtr=POINTER TO TimeVal;
27  IOTimer=RECORD
28   node:IORequest;
29   time:TimeVal
30  END;
31  IOTimerPtr=POINTER TO IOTimer;
32
33 PROCEDURE AddTime(timer{14}:ADDRESS;
34                   dest{8},source{9}:TimeValPtr); CODE -42;
35 PROCEDURE CmpTime(timer{14}:ADDRESS;
36                   tv1{8},
37                   tv2{9}:TimeValPtr):INTEGER; CODE -54;
38 PROCEDURE SubTime(timer{14}:ADDRESS;
39                   dest{8},source{9}:TimeValPtr); CODE -48;
40
41 END Timer.
42
```

TrackDisk

```
(*$M-*)
DEFINITION MODULE TrackDisk;

FROM SYSTEM IMPORT
 BYTE,SHIFT;
FROM Exec IMPORT
 clear,read,update,write,nonstd,IOStdReq,Unit;

CONST
 trackDiskName="trackdisk.device";
 motor=nonstd;
 seek=nonstd+1;
 format=nonstd+2;
 remove=nonstd+3;
 changeNum=nonstd+4;
 changeState=nonstd+5;
 protStatus=nonstd+6;
 rawRead=nonstd+7;
 rawWrite=nonstd+8;
 getDriveType=nonstd+9;
 getNumTracks=nonstd+10;
 addChangeInt=nonstd+11;
 remChangeInt=nonstd+12;
 lastComm=nonstd+13;
 extCom=8000H;
 extWrite=write+extCom;
 extRead=read+extCom;
 extMotor=motor+extCom;
 extSeek=seek+extCom;
 extFormat=format+extCom;
 extUpdate=update+extCom;
 extClear=clear+extCom;
 extRawRead=rawRead+extCom;
 extRawWrite=rawWrite+extCom;
 numSecs=11;
 numUnits=4;
 sector=512;
 secShift=9;
 labelSize=16;
 indexSync=4;
 allowNon35=0;
 drive35=1;
 drive525=2;
 notSpecified=20;
 noSecHdr=21;
 badSecPreamble=22;
 badSecId=23;
 badHdrSum=24;
 badSecSum=25;
 tooFewSecs=26;
 badSecHdr=27;
 writeProt=28;
 diskChanged=29;
```

```
 1  seekError=30;
 2  noMem=31;
 3  badUnitNum=32;
 4  badDriveType=33;
 5  driveInUse=34;
 6  postReset=35;
 7
 8 TYPE
 9  IOTrackDisk=RECORD
10   req:IOStdReq;
11   count:LONGCARD;
12   secLabel:LONGCARD;
13  END;
14  IOTrackDiskPtr=POINTER TO IOTrackDisk;
15  TDUPublicUnit=RECORD
16   unit:Unit;
17   comp01Track:CARDINAL;
18   comp10Track:CARDINAL;
19   comp11Track:CARDINAL;
20   stepDelay:LONGCARD;
21   settleDelay:LONGCARD;
22   retryCnt:[0..255]
23  END;
24
25 END TrackDisk.
26
```

Translator

```
 5 DEFINITION MODULE Translator {"translator.library", 33};
 6
 7 FROM SYSTEM IMPORT
 8  ADDRESS;
 9
10 CONST
11  notUsed=-1;
12  noMem=-2;
13  makeBad=-4;
14
15 PROCEDURE Translate(in{8}:ADDRESS;
16                      inLen{0}:LONGINT;
17                      out{9}:ADDRESS;
18                      outLen{1}:LONGINT): LONGINT; CODE -30;
19
20 END Translator.
21
```

Workbench

```
(*$M-*)
DEFINITION MODULE Workbench;

FROM SYSTEM IMPORT
 ADDRESS;
FROM Intuition IMPORT
 Gadget,GadgetFlags,GadgetFlagSet,NewWindow;
FROM Exec IMPORT
 Message,MsgPortPtr,List;

CONST
 diskMagic=0E310H;
 diskVersion=1;
 gadgetBackFill=GadgetFlagSet{gadgHBox};
 noIconPosition=MIN(LONGINT);

TYPE
 WBObjectType=(
  wb0,disk,drawer,tool,project,garbage,device,kick
 );
 MType=(mt0,pstd,toolExit,diskChange,timer,closeDown,ioProc);
 DiskObjectPtr=POINTER TO DiskObject;
 DrawerDataPtr=POINTER TO DrawerData;
 FreeListPtr=POINTER TO FreeList;
 WBArgPtr=POINTER TO WBArg;
 WBStartupPtr=POINTER TO WBStartup;
 WBArg=RECORD
  lock:ADDRESS;
  name:ADDRESS
 END;
 WBArgumentsPtr=POINTER TO ARRAY [0..255] OF WBArg;
 WBStartup=RECORD
  message:Message;
  process:MsgPortPtr;
  segment:ADDRESS;
  numArgs:LONGINT;
  toolWindow:ADDRESS;
  argList:WBArgumentsPtr;
 END;
 FreeList=RECORD
  numFree:INTEGER;
  memList:List
 END;
 DiskObject=RECORD
  magic:CARDINAL;
  version:CARDINAL;
  gadget:Gadget;
  type:WBObjectType;
  defaultTool:ADDRESS;
  toolTypes:ADDRESS;
  currentX:LONGINT;
  currentY:LONGINT;
  drawerData:DrawerDataPtr;
```

```
1    toolWindow:ADDRESS;
2    stackSize:LONGINT
3   END;
4   DrawerData=RECORD
5    newWindow:NewWindow;
6    currentX:LONGINT;
7    currentY:LONGINT;
8   END;
9
10 CONST
11  drawerDataFileSize=SIZE(DrawerData);
12
13 END Workbench.
14
```

14 Cross-Reference

Auf den folgenden Seiten befindet sich einen alphabetischen Index aller in den Kapiteln 12 und 13 deklarierten Bezeichner. Zu jedem Bezeichner sind Modulname, Seiten- und Zeilennummer (nur für Kapitel 13) angegeben:

```
Execute ..... Dos.....42 13-17
```

Die Deklaration von Execute befindet sich demnach in der Beschreibung des Moduls Dos auf Seite 13-17 auf der Zeile 42.

a0 ... Assembler ... 12-12
a1 ... Assembler ... 12-12
a15 ... DiskFont ... 25 13-10
a2 ... Assembler ... 12-12
a3 ... Assembler ... 12-12
a4 ... Assembler ... 12-12
a5 ... Assembler ... 12-12
a6 ... Assembler ... 12-12
a7 ... Assembler ... 12-12
abc ... Hardware ... 55 13-50
abcdB ... Assembler ... 12-12
abcdmB ... Assembler ... 12-12
AbleICR ... Resources ... 23 13-92
abnc ... Hardware ... 55 13-50
abort ... Parallel ... 27 13-86
abort ... Serial ... 21 13-95
abortIO ... Exec ... 38 13-22
AbortIO ... Exec ... 55 13-24
AbortIO ... ExecSupport ... 15 13-29
absJoystick ... GamePort ... 30 13-34
absL ... Assembler ... 12-12
absW ... Assembler ... 12-12
accessRead ... Dos ... 32 13-12
accessWrite ... Dos ... 33 13-12
ack ... ASCII ... 12-11
Acos ... MathIEEEDoubTrans ... 8 13-82
Acos ... MathTrans ... 9 13-83
actionNotKnown ... Dos ... 46 13-13
activate ... Intuition ... 36 13-64
ActivateGadget ... Intuition ... 28 13-72
ActivateWindow ... Intuition ... 32 13-72
ActivationFlags ... Intuition ... 4 13-61
ActivationFlagSet ... Intuition ... 9 13-61
active ... Exec ... 15 13-22
active ... Parallel ... 27 13-86
active ... Resources ... 46 13-92
active ... Serial ... 20 13-95
activewindow ... InputEvent ... 18 13-57
activeWindow ... Intuition ... 36 13-63
addaL ... Assembler ... 12-13
AddAnimOb ... Graphics ... 12 13-45
addaW ... Assembler ... 12-13
addB ... Assembler ... 12-12
AddBob ... Graphics ... 15 13-45
addChangeInt ... TrackDisk ... 26 13-98
AddConfigDev ... Expansion ... 12 13-32
AddDevice ... Exec ... 56 13-24
AddDosNode ... Expansion ... 13 13-32
added ... Exec ... 9 13-21
AddFont ... Graphics ... 17 13-45
AddFreeList ... Icon ... 14 13-55
AddGadget ... Intuition ... 33 13-72
AddGList ... Intuition ... 36 13-72
addHandler ... Input ... 13 13-56
AddHead ... Exec ... 57 13-24
addiB ... Assembler ... 12-13
AddICRVector ... Resources ... 26 13-92
addiL ... Assembler ... 12-13
AddIntServer ... Exec ... 2 13-25
additive ... PrtBase ... 17 13-91
addiW ... Assembler ... 12-13
addL ... Assembler ... 12-12
AddLibrary ... Exec ... 4 13-25
addmB ... Assembler ... 12-13
AddMemList ... Exec ... 5 13-25
addmL ... Assembler ... 12-13
addmW ... Assembler ... 12-13
AddPort ... Exec ... 10 13-25
addqB ... Assembler ... 12-13
addqL ... Assembler ... 12-13
addqW ... Assembler ... 12-13
addRequest ... Timer ... 15 13-97
addResetHandler ... Keyboard ... 15 13-77
AddResource ... Exec ... 11 13-25

AddSemaphore ..... Exec 12 13-25
AddTail ..... Exec 14 13-25
AddTask ..... Exec 16 13-25
AddTime ..... Timer 33 13-97
AddVSprite ..... Graphics 18 13-45
addW ..... Assembler 12-12
addxB ..... Assembler 12-13
addxL ..... Assembler 12-13
addxmB ..... Assembler 12-13
addxmL ..... Assembler 12-13
addxmW ..... Assembler 12-13
addxW ..... Assembler 12-13
adec ..... Assembler 12-12
adir ..... Assembler 12-12
AdkFlags ..... Hardware 16 13-50
AdkFlagSet ..... Hardware 21 13-50
adkSet ..... Hardware 20 13-50
af10 ..... DiskFont 24 13-10
af11 ..... DiskFont 24 13-10
af12 ..... DiskFont 24 13-10
af13 ..... DiskFont 24 13-10
af14 ..... DiskFont 24 13-10
af2 ..... DiskFont 23 13-10
af2 ..... Exec 51 13-23
af3 ..... DiskFont 23 13-10
af3 ..... Exec 51 13-23
af4 ..... DiskFont 23 13-10
af5 ..... DiskFont 23 13-10
af5 ..... Exec 51 13-23
af6 ..... DiskFont 23 13-10
af6 ..... Exec 51 13-23
af7 ..... DiskFont 23 13-10
af7 ..... Exec 51 13-23
af8 ..... DiskFont 24 13-10
af9 ..... DiskFont 24 13-10
aidr ..... Assembler 12-12
aidx ..... Assembler 12-12
ainc ..... Assembler 12-12
Alert ..... Exec 19 13-25
alertLayersNoMem ..... Graphics 9 13-42
Alloc ..... Heap 12-33
alloc0 ..... Resources 45 13-92
alloc1 ..... Resources 45 13-92
alloc2 ..... Resources 45 13-92
alloc3 ..... Resources 45 13-92
AllocAbs ..... Exec 21 13-25
allocate ..... Audio 25 13- 3
Allocate ..... Exec 23 13-25
Allocate ..... Heap 12-33
ALLOCATE ..... Storage 12-52
allocated ..... GamePort 18 13-34
AllocBoardMem ..... Expansion 17 13-32
AllocConfigDev ..... Expansion 19 13-32
AllocEntry ..... Exec 25 13-25
AllocExpansionMem ..... Expansion 20 13-32
allocFailed ..... Audio 31 13- 3
allocMaxprec ..... Audio 17 13- 3
AllocMem ..... Exec 27 13-25
AllocMem ..... Heap 12-33
allocMinprec ..... Audio 16 13- 3
AllocMiscResource ..... Resources 39 13-93
AllocPotBits ..... Resources 56 13-93
AllocRaster ..... Graphics 20 13-45
AllocRemember ..... Intuition 42 13-72
AllocSignal ..... Exec 30 13-25
AllocTrap ..... Exec 32 13-25
AllocUnit ..... Resources 13 13-93
AllocWBObject ..... Icon 24 13-55
allowNon35 ..... TrackDisk 45 13-98
AlohaWorkbench ..... Intuition 46 13-72
alphaP101 ..... Intuition 12 13-67
alrm ..... Hardware 33 13-53
alt ..... KeyMap 14 13-78
altKeyMap ..... Intuition 7 13-61

altLeft	Intuition	11	13-64
altRight	Intuition	12	13-64
amiga	Resources	9	13-93
amigaKeys	Intuition	15	13-64
amigaLeft	Intuition	13	13-64
amigaRight	Intuition	14	13-64
anbc	Hardware	55	13-50
anbnc	Hardware	55	13-50
andB	Assembler		12-13
andiB	Assembler		12-13
andiCCR	Assembler		12-13
andiL	Assembler		12-13
andiSR	Assembler		12-13
andiW	Assembler		12-13
andL	Assembler		12-13
andmB	Assembler		12-13
andmL	Assembler		12-13
andmW	Assembler		12-13
AndRectRegion	Graphics	22	13-45
AndRegionRegion	Graphics	24	13-45
andW	Assembler		12-13
anfracsize	Graphics	1	13-39
Animate	Graphics	27	13-45
AnimComp	Graphics	50	13-39
AnimCompPtr	Graphics	16	13-36
animhalf	Graphics	2	13-39
AnimOb	Graphics	7	13-40
AnimObPtr	Graphics	17	13-36
aoff	Assembler		12-12
aORb	Hardware	4	13-51
aORc	Hardware	5	13-51
archive	Dos	48	13-12
arctan	MathLib0		12-42
arctan	MathLibFFP		12-44
arctan	MathLibLong		12-46
AreaCircle	GfxMacros	33	13-35
AreaDraw	Graphics	29	13-45
AreaEllipse	Graphics	32	13-45
AreaEnd	Graphics	37	13-45
AreaInfo	Graphics	12	13-42
AreaInfoPtr	Graphics	18	13-36
AreaMove	Graphics	38	13-45
areaOutline	Graphics	51	13-42
arranging	Windows		12-64
asciiDel	InputEvent	32	13-57
asciiFirst	InputEvent	30	13-57
asciiLast	InputEvent	31	13-57
ash1	Hardware	56	13-50
ash2	Hardware	56	13-50
ash4	Hardware	56	13-50
ash8	Hardware	57	13-50
Asin	MathIEEEDoubTrans	9	13-82
Asin	MathTrans	10	13-83
askCType	GamePort	14	13-34
askDefaultKeyMap	Console	19	13- 6
AskFont	Graphics	41	13-45
askKeyMap	Console	17	13- 6
AskSoftStyle	Graphics	43	13-45
askTrigger	GamePort	16	13-34
aslB	Assembler		12-13
asliB	Assembler		12-13
asliL	Assembler		12-13
asliW	Assembler		12-13
aslL	Assembler		12-13
aslmW	Assembler		12-13
aslW	Assembler		12-13
aspect	Printer	14	13-89
aspectHoriz	Intuition	26	13-68
aspectVert	Intuition	27	13-68
asrB	Assembler		12-13
asriB	Assembler		12-13
asriL	Assembler		12-13
asriW	Assembler		12-13
asrL	Assembler		12-13

asrmW	Assembler	12-13
asrW	Assembler	12-13
Assert	Arts	12- 9
assertion	Arts	12- 7
Atan	MathIEEEDoubTrans 10	13-82
Atan	MathTrans 11	13-83
aTOd	Hardware 7	13-51
AttemptLockLayerRom	Graphics 45	13-45
AttemptSemaphore	Exec 33	13-25
AttnFlags	Exec 50	13-23
AttnFlagSet	Exec 54	13-23
aud0	Hardware 31	13-50
aud0i	Hardware 41	13-50
aud1	Hardware 31	13-50
aud1i	Hardware 42	13-50
aud2	Hardware 31	13-50
aud2i	Hardware 42	13-50
aud3	Hardware 31	13-50
aud3i	Hardware 42	13-50
audioName	Audio 14	13- 3
aul	Hardware 18	13-51
autoBackPen	Intuition 48	13-62
autoDrawMode	Intuition 49	13-62
autoFrontPen	Intuition 47	13-62
autoinit	Exec 30	13-23
autoITextFont	Intuition 52	13-62
autoKnob	Intuition 4	13-62
autoLeftEdge	Intuition 50	13-62
autoNextText	Intuition 53	13-62
AutoRequest	Intuition 47	13-72
autoTopEdge	Intuition 51	13-62
Available	Heap	12-33
Available	Storage	12-52
AvailFont	DiskFont 47	13-10
AvailFontHeader	DiskFont 51	13-10
AvailFontHeaderPtr	DiskFont 55	13-10
AvailFonts	DiskFont 57	13-10
AvailFontTypes	DiskFont 22	13-10
AvailFontTypeSet	DiskFont 26	13-10
AvailMem	Exec 36	13-25
aXORc	Hardware 6	13-51
b2Bobber	Graphics 6	13-39
b2Norm	Graphics 4	13-39
b2Swap	Graphics 5	13-39
backDrop	Intuition 35	13-64
backSaved	Graphics 9	13-39
badDimension	Printer 52	13-88
badDriveType	TrackDisk 4	13-99
badHdrSum	TrackDisk 52	13-98
badSecHdr	TrackDisk 55	13-98
badSecId	TrackDisk 51	13-98
badSecPreamble	TrackDisk 50	13-98
badSecSum	TrackDisk 53	13-98
badStreamName	Dos 44	13-13
badUnitNum	TrackDisk 3	13-99
base	Exec 44	13-21
baud110	Intuition 9	13-68
baud1200	Intuition 11	13-68
baud19200	Intuition 15	13-68
baud2400	Intuition 12	13-68
baud300	Intuition 10	13-68
baud4800	Intuition 13	13-68
baud9600	Intuition 14	13-68
baudMidi	Intuition 16	13-68
baudMismatch	Serial 42	13-95
BC0Flags	Hardware 54	13-50
BC0FlagSet	Hardware 1	13-51
BC1Flags	Hardware 10	13-51
BC1FlagSet	Hardware 14	13-51
bcc	Assembler	12-13
bchg	Assembler	12-13
bchgi	Assembler	12-13
bclr	Assembler	12-13
bclri	Assembler	12-13

bcs ..... Assembler 12-13
bDrawn ..... Graphics 37 13-39
beeping ..... Intuition 43 13-64
beginIO ..... Exec 39 13-22
BeginIO ..... ExecSupport 14 13-29
beginning ..... Dos 23 13-12
BeginRefresh ..... Intuition 55 13-72
BeginUpdate ..... Layers 13 13-80
BehindLayer ..... Layers 14 13-80
bel ..... ASCII 12-11
beq ..... Assembler 12-13
bf10 ..... Hardware 12 13-51
bf11 ..... Hardware 12 13-51
bf2 ..... Graphics 36 13-39
bf3 ..... Graphics 36 13-39
bf4 ..... Graphics 36 13-39
bf5 ..... Graphics 36 13-39
bf6 ..... Graphics 36 13-39
bf7 ..... Graphics 36 13-39
bf7 ..... Hardware 12 13-51
bf8 ..... Hardware 12 13-51
bf9 ..... Hardware 12 13-51
bge ..... Assembler 12-13
bgr ..... PrtBase 19 13-91
bgrw ..... PrtBase 21 13-91
bgrWb ..... PrtBase 20 13-91
bgt ..... Assembler 12-13
bhi ..... Assembler 12-13
bindTime ..... Expansion 38 13-31
BitMap ..... Graphics 33 13-40
BitMapPtr ..... Graphics 19 13-36
BitTable ..... KeyMap 27 13-78
BitTablePtr ..... KeyMap 29 13-78
black ..... Console 26 13- 6
blackBg ..... Console 35 13- 6
ble ..... Assembler 12-13
blit ..... Hardware 41 13-50
blithog ..... Hardware 32 13-50
blitMsgFault ..... Graphics 32 13-41
blitReverse ..... Hardware 17 13-51
blitter ..... Hardware 31 13-50
bls ..... Assembler 12-13
blt ..... Assembler 12-13
BltBitMap ..... Graphics 47 13-45
BltBitMapRastPort ..... Graphics 1 13-46
BltClear ..... Graphics 10 13-46
bltdone ..... Hardware 32 13-50
BltMaskBitMapRastPort ..... Graphics 13 13-46
Bltnode ..... Hardware 32 13-51
BltnodePtr ..... Hardware 31 13-51
bltnzero ..... Hardware 32 13-50
BltPattern ..... Graphics 24 13-46
BltTemplate ..... Graphics 31 13-46
blue ..... Console 30 13- 6
blueBg ..... Console 39 13- 6
bmi ..... Assembler 12-13
bms ..... Printer 35 13-88
BndryOff ..... GfxMacros 27 13-35
bne ..... Assembler 12-13
Bob ..... Graphics 40 13-39
BobFlags ..... Graphics 35 13-39
BobFlagSet ..... Graphics 39 13-39
bobIsComp ..... Graphics 36 13-39
bobNix ..... Graphics 37 13-39
BobPtr ..... Graphics 20 13-36
bobsAway ..... Graphics 37 13-39
bobUpdate ..... Graphics 10 13-39
bold ..... Console 22 13- 6
bold ..... Graphics 43 13-42
boolExtend ..... Intuition 8 13-61
boolGadget ..... Intuition 14 13-61
BoolInfo ..... Intuition 55 13-61
boolMask ..... Intuition 52 13-61
BootBlock ..... BootBlock 11 13- 4

bootSects ..... BootBlock 18 13- 4
bootTime ..... Expansion 35 13-31
Border ..... Intuition 9 13-63
borderHit ..... Graphics 41 13-37
borderless ..... Intuition 36 13-64
BorderPtr ..... Intuition 26 13-59
bottomBorder ..... Intuition 6 13-61
bottomHit ..... Graphics 43 13-37
bpl ..... Assembler 12-13
bra ..... Assembler 12-13
break ..... Serial 16 13-95
breakPoint ..... Arts 12- 7
BreakPoint ..... Arts 12- 9
brother15XL ..... Intuition 12 13-67
bs ..... ASCII 12-11
bset ..... Assembler 12-13
bseti ..... Assembler 12-13
bsh1 ..... Hardware 12 13-51
bsh2 ..... Hardware 12 13-51
bsh4 ..... Hardware 12 13-51
bsh8 ..... Hardware 13 13-51
bsr ..... Assembler 12-13
BSTR ..... Dos 38 13-12
btst ..... Assembler 12-13
btsti ..... Assembler 12-13
buf1024 ..... Intuition 43 13-68
buf16000 ..... Intuition 47 13-68
buf2048 ..... Intuition 44 13-68
buf4096 ..... Intuition 45 13-68
buf512 ..... Intuition 42 13-68
buf8000 ..... Intuition 46 13-68
bufErr ..... Serial 42 13-95
bufferMemory ..... Printer 54 13-88
bufOverflow ..... Serial 43 13-95
bufrRead ..... Serial 23 13-95
bufSizeBits ..... Intuition 41 13-68
bufTooBig ..... Parallel 22 13-86
BuildSysRequest ..... Intuition 56 13-72
BumpRevision ..... Icon 12 13-55
busWidth ..... Expansion 31 13-31
busy ..... Serial 36 13-95
BusyRead ..... Terminal 12-59
bvc ..... Assembler 12-13
bvs ..... Assembler 12-13
bw ..... PrtBase 12 13-91
bWaiting ..... Graphics 37 13-39
bwAlpha ..... PrtBase 9 13-91
bwGfx ..... PrtBase 10 13-91
Byte ..... Exec 13 13-19
Byte ..... Hardware 12 13-50
byteWide ..... Expansion 33 13-31
c0First ..... InputEvent 28 13-57
c0Last ..... InputEvent 29 13-57
c1First ..... InputEvent 33 13-57
c1Last ..... InputEvent 34 13-57
c23 ..... Intuition 37 13-63
c24 ..... Intuition 37 13-63
c25 ..... Intuition 37 13-63
c26 ..... Intuition 37 13-63
c27 ..... Intuition 37 13-63
c28 ..... Intuition 37 13-63
c29 ..... Intuition 38 13-63
c30 ..... Intuition 38 13-63
Call ..... Arts 12- 9
cam ..... Printer 38 13-88
can ..... ASCII 12-11
cancel ..... Printer 52 13-88
cantAlloc ..... Narrator 19 13-84
capsLock ..... InputEvent 47 13-57
CapString ..... Str 12-53
carrierDetect ..... Serial 36 13-95
Cause ..... Exec 38 13-25
cbit ..... Assembler 12-12
cbmMps1000 ..... Intuition 12 13-67

CBump	Graphics	39	13-46
CDInputHandler	Console	13	13- 7
CEND	GfxMacros	31	13-35
center	Printer	13	13-89
chainedConfig	Expansion	5	13-31
changed	Exec	53	13-21
changeNum	TrackDisk	19	13-98
ChangeSprite	Graphics	40	13-46
changeState	TrackDisk	20	13-98
channelStolen	Audio	32	13- 3
CHARPtr	FileSystem		12-28
checked	Intuition	5	13-60
CheckIO	Exec	39	13-25
checkIt	Intuition	4	13-60
checkWidth	Intuition	12	13-60
chip	Exec	6	13-20
chkW	Assembler		12-13
CIAA	Hardware	11	13-54
ciaa	Hardware	49	13-54
ciaaName	Resources	17	13-92
CiaaPraFlags	Hardware	48	13-53
CiaaPraFlagSet	Hardware	52	13-53
CiaaPrbFlags	Hardware	53	13-53
CiaaPrbFlagSet	Hardware	54	13-53
CIAB	Hardware	29	13-54
ciab	Hardware	50	13-54
ciabName	Resources	18	13-92
CiabPraFlags	Hardware	55	13-53
CiabPraFlagSet	Hardware	1	13-54
CiabPrbFlags	Hardware	2	13-54
CiabPrbFlagSet	Hardware	6	13-54
CiaCraFlags	Hardware	38	13-53
CiaCraFlagSet	Hardware	42	13-53
CiaCrbFlags	Hardware	43	13-53
CiaCrbFlagSet	Hardware	47	13-53
CiaIcrFlags	Hardware	33	13-53
CiaIcrFlagSet	Hardware	34	13-53
CiaResourcePtr	Resources	21	13-92
CINIT	GfxMacros	28	13-35
cl5	InputEvent	15	13-57
Class	InputEvent	14	13-57
classMax	InputEvent	22	13-57
cleanme	Hardware	43	13-51
cleanup	Hardware	42	13-51
clear	Exec	32	13-22
Clear	Windows		12-64
ClearDMRequest	Intuition	7	13-73
ClearEOL	Graphics	43	13-46
ClearMenuStrip	Intuition	9	13-73
ClearPointer	Intuition	10	13-73
ClearRectRegion	Graphics	44	13-46
ClearRegion	Graphics	47	13-46
ClearScreen	Graphics	48	13-46
clearToSend	Serial	36	13-95
ClipBlit	Graphics	49	13-46
clipboardName	Clipboard	14	13- 5
ClipboardUnitPartial	Clipboard	21	13- 5
ClipboardUnitPartialPtr	Clipboard	25	13- 5
ClipRect	Graphics	23	13-37
ClipRectPtr	Graphics	21	13-36
Close	Dos	25	13-17
close	Exec	49	13-21
Close	FileSystem		12-28
close	Intuition	25	13-61
CloseDevice	Exec	40	13-25
closeDown	Workbench	25	13-101
CloseFont	Graphics	1	13-47
closeGadget	Intuition	17	13-69
CloseInput	InOut		12-36
CloseLibrary	Exec	41	13-25
CloseOutput	InOut		12-36
CloseScreen	Intuition	11	13-73
closewindow	InputEvent	16	13-57
CloseWindow	Intuition	12	13-73

closeWindow ..... Intuition 34 13-63
CloseWindow ..... Windows 12-64
CloseWorkBench ..... Intuition 13 13-73
closing ..... Windows 12-64
clr0 ..... Console 44 13- 6
clr0Bg ..... Console 52 13- 6
clr1 ..... Console 45 13- 6
clr1Bg ..... Console 53 13- 6
clr2 ..... Console 46 13- 6
clr2Bg ..... Console 54 13- 6
clr3 ..... Console 47 13- 6
clr3Bg ..... Console 55 13- 6
clr4 ..... Console 48 13- 6
clr4Bg ..... Console 56 13- 6
clr5 ..... Console 49 13- 6
clr5Bg ..... Console 57 13- 6
clr6 ..... Console 50 13- 6
clr6Bg ..... Console 1 13- 7
clr7 ..... Console 51 13- 6
clr7Bg ..... Console 2 13- 7
clrB ..... Assembler 12-13
clrL ..... Assembler 12-13
clrW ..... Assembler 12-13
CMOVE ..... GfxMacros 29 13-35
CMove ..... Graphics 2 13-47
cmpaL ..... Assembler 12-13
cmpaW ..... Assembler 12-13
cmpB ..... Assembler 12-13
cmpiB ..... Assembler 12-13
cmpiL ..... Assembler 12-13
cmpiW ..... Assembler 12-13
cmpL ..... Assembler 12-13
cmpmB ..... Assembler 12-13
cmpmL ..... Assembler 12-13
cmpmW ..... Assembler 12-13
CmpTime ..... Timer 35 13-97
cmpW ..... Assembler 12-13
coldstart ..... Exec 30 13-23
CollTable ..... Graphics 30 13-40
CollTablePtr ..... Graphics 22 13-36
color ..... PrtBase 5 13-91
colorGfx ..... PrtBase 11 13-91
ColorMap ..... Graphics 22 13-44
ColorMapPtr ..... Graphics 23 13-36
colorOn ..... Graphics 41 13-38
comCD ..... Hardware 56 13-53
comCTS ..... Hardware 56 13-53
comDSR ..... Hardware 56 13-53
comDTR ..... Hardware 57 13-53
CommandLineInterface ..... Dos 33 13-16
CommandLineInterfacePtr ..... Dos 23 13-14
commCodeFirst ..... InputEvent 26 13-57
commCodeLast ..... InputEvent 27 13-57
commentTooBig ..... Dos 57 13-13
commSeq ..... Intuition 4 13-60
commWidth ..... Intuition 13 13-60
Compare ..... Str 12-53
Compare ..... Strings 12-56
complement ..... Graphics 41 13-42
comRTS ..... Hardware 56 13-53
Concat ..... Str 12-53
ConfigBoard ..... Expansion 24 13-32
ConfigChain ..... Expansion 27 13-32
ConfigDev ..... Expansion 42 13-31
ConfigDevPtr ..... Expansion 41 13-31
configMe ..... Expansion 1 13-32
configTime ..... Expansion 37 13-31
consoleName ..... Console 16 13- 6
control ..... InputEvent 47 13-57
control ..... KeyMap 14 13-78
Controller ..... GamePort 30 13-34
ConUnit ..... ConUnit 29 13- 8
ConUnitPtr ..... ConUnit 9 13- 9
Coord ..... Hardware 46 13-51

coper	Hardware	41	13-50
Copinit	Graphics	32	13-38
CopinitPtr	Graphics	24	13-36
CopIns	Graphics	55	13-37
CopInsPtr	Graphics	25	13-36
CopList	Graphics	15	13-38
CopListPtr	Graphics	26	13-36
copper	Hardware	31	13-50
Copy	Str		12-53
Copy	Strings		12-56
copyDir	Dos	38	13-15
CopyMem	Exec	42	13-25
CopyMemQuick	Exec	45	13-25
CopyPos	Str		12-53
CopySBitMap	Graphics	5	13-47
Cos	MathIEEEDoubTrans	11	13-82
cos	MathLib0		12-42
cos	MathLibFFP		12-44
cos	MathLibLong		12-46
Cos	MathTrans	12	13-83
Cosh	MathIEEEDoubTrans	12	13-82
Cosh	MathTrans	13	13-83
Cprlist	Graphics	10	13-38
CprlistPtr	Graphics	27	13-36
cr	ASCII		12-11
craInmode	Hardware	39	13-53
craLoad	Hardware	39	13-53
craOutmode	Hardware	39	13-53
craPbon	Hardware	39	13-53
craRunmode	Hardware	39	13-53
craSpmode	Hardware	40	13-53
craStart	Hardware	39	13-53
craTodin	Hardware	41	13-53
crbAlarm	Hardware	46	13-53
crbInmode0	Hardware	44	13-53
crbInmode1	Hardware	45	13-53
crbLoad	Hardware	44	13-53
crbOutmode	Hardware	44	13-53
crbPbon	Hardware	44	13-53
crbRunmode	Hardware	44	13-53
crbStart	Hardware	44	13-53
CreateBehindLayer	Layers	15	13-80
CreateDir	Dos	26	13-17
createDir	Dos	41	13-15
CreateExtIO	ExecSupport	19	13-29
CreatePort	ExecSupport	16	13-29
CreateProc	Dos	27	13-17
CreateStdIO	ExecSupport	22	13-29
CreateTask	ExecSupport	24	13-29
CreateUpfrontLayer	Layers	24	13-80
csi	ASCII		12-11
ctcHClrTab	Console	5	13- 7
ctcHClrTabsAll	Console	6	13- 7
ctcHSetTab	Console	4	13- 7
ctrlC	Dos	17	13-14
ctrlD	Dos	18	13-14
ctrlE	Dos	19	13-14
ctrlF	Dos	20	13-14
current	Dos	24	13-12
CurrentBinding	Expansion	4	13-32
CurrentBindingPtr	Expansion	10	13-32
CurrentDir	Dos	32	13-17
CurrentLevel	Arts		12- 9
currentReadId	Clipboard	16	13- 5
CurrentTime	Intuition	14	13-73
currentVolume	Dos	29	13-15
currentWriteId	Clipboard	17	13- 5
cursorDown	Intuition	56	13-63
cursorLeft	Intuition	1	13-64
cursorRight	Intuition	57	13-63
cursorUp	Intuition	55	13-63
custom	Hardware	30	13-53
Custom	Hardware	49	13-51
custom	Intuition	35	13-68

customBitMap ..... Intuition 43 13-64
customName ..... Intuition 12 13-67
customScreen ..... Intuition 51 13-64
CWAIT ..... GfxMacros 30 13-35
CWait ..... Graphics 6 13-47
cyan ..... Console 32 13- 6
cyanBg ..... Console 41 13- 6
d0 ..... Assembler 12-12
d1 ..... Assembler 12-12
d2 ..... Assembler 12-12
d3 ..... Assembler 12-12
d4 ..... Assembler 12-12
d5 ..... Assembler 12-12
d6 ..... Assembler 12-12
d7 ..... Assembler 12-12
dataSetReady ..... Serial 36 13-95
dataTerminalReady ..... Serial 37 13-95
Date ..... Dos 41 13-12
DatePtr ..... Dos 46 13-12
DateStamp ..... Dos 34 13-17
datlx ..... Resources 52 13-93
datly ..... Resources 52 13-93
datrx ..... Resources 52 13-93
datry ..... Resources 52 13-93
dbcc ..... Assembler 12-13
dbcs ..... Assembler 12-13
dbeq ..... Assembler 12-13
dbf ..... Assembler 12-13
dbge ..... Assembler 12-13
dbgt ..... Assembler 12-13
dbhi ..... Assembler 12-13
dble ..... Assembler 12-13
dblpf ..... Graphics 42 13-38
dbls ..... Assembler 12-13
dblt ..... Assembler 12-13
dbmi ..... Assembler 12-13
dbne ..... Assembler 12-13
dbpl ..... Assembler 12-13
dbra ..... Assembler 12-13
dbt ..... Assembler 12-13
dBuffer ..... Graphics 51 13-42
DBufPacket ..... Graphics 24 13-40
DBufPacketPtr ..... Graphics 28 13-36
dbvc ..... Assembler 12-13
dbvs ..... Assembler 12-13
dc1 ..... ASCII 12-11
dc2 ..... ASCII 12-11
dc3 ..... ASCII 12-11
dc4 ..... ASCII 12-11
ddir ..... Assembler 12-12
dead ..... KeyMap 14 13-78
deadendAlert ..... Intuition 57 13-68
DeadPrefixBytes ..... KeyMap 16 13-78
DeadPrefixByteSet ..... KeyMap 17 13-78
Deallocate ..... Exec 48 13-25
Deallocate ..... Heap 12-33
DEALLOCATE ..... Storage 12-52
Debug ..... Exec 51 13-25
DebugProcedure ..... Arts 12- 9
default ..... Console 34 13- 6
defaultBg ..... Console 43 13- 6
deferRefresh ..... Intuition 35 13-60
defFreq ..... Narrator 39 13-84
defMode ..... Narrator 41 13-84
defPitch ..... Narrator 36 13-84
defRate ..... Narrator 37 13-84
defSex ..... Narrator 40 13-84
defVol ..... Narrator 38 13-84
del ..... ASCII 12-11
Delay ..... Dos 35 13-17
delete ..... Dos 48 13-12
Delete ..... FileSystem 12-29
Delete ..... Strings 12-56
DeleteExtIO ..... ExecSupport 21 13-29

DeleteFile	Dos	36	13-17
DeleteLayer	Layers	33	13-80
deleteObject	Dos	35	13-15
DeletePort	ExecSupport	18	13-29
deleteProtected	Dos	2	13-14
deleteProtected	FileSystem		12-28
DeleteStdIO	ExecSupport	23	13-29
DeleteTask	ExecSupport	28	13-29
delExp	Exec	53	13-21
deltaMove	Intuition	37	13-63
den1	Printer	50	13-87
den2	Printer	49	13-87
den3	Printer	48	13-87
den4	Printer	47	13-87
den5	Printer	46	13-87
den6	Printer	45	13-87
density1	Printer	14	13-89
density2	Printer	14	13-89
density3	Printer	19	13-89
density4	Printer	15	13-89
densityMask	Printer	20	13-89
desc	Hardware	11	13-51
designed	Graphics	47	13-42
dest	Hardware	56	13-50
DetectCtrlC	Arts		12- 9
detectedBreak	Serial	45	13-95
devBusy	Parallel	22	13-86
devBusy	Serial	42	13-95
device	Dos	51	13-16
Device	Exec	11	13-22
device	Exec	16	13-19
device	Workbench	23	13-101
DeviceData	PrtBase	22	13-90
DeviceDataPtr	PrtBase	30	13-90
DeviceList	Dos	52	13-16
DeviceListPtr	Dos	15	13-16
DeviceListType	Dos	51	13-16
deviceNotMounted	Dos	55	13-13
deviceNotMounted	FileSystem		12-28
DeviceProc	Dos	37	13-17
DevicePtr	Exec	14	13-22
df11	Hardware	32	13-50
df12	Hardware	32	13-50
dfhId	DiskFont	19	13-10
dftchMask	Graphics	54	13-38
diab630	Intuition	12	13-67
diabAdvD25	Intuition	13	13-67
diabC150	Intuition	13	13-67
DiagArea	Expansion	19	13-31
diagValid	Expansion	6	13-31
die	Dos	27	13-15
dimensionOvflow	Printer	52	13-88
directory	Dos	51	13-16
directoryNotEmpty	Dos	53	13-13
dirNotFound	Dos	42	13-13
Disable	Exec	52	13-25
DiscResource	Resources	48	13-92
DiscResourceFlags	Resources	44	13-92
DiscResourceFlagSet	Resources	47	13-92
DiscResourcePtr	Resources	4	13-93
DiscResourceUnit	Resources	37	13-92
DiscResourceUnitPtr	Resources	43	13-92
disk	DiskFont	23	13-10
disk	Hardware	31	13-50
disk	Workbench	23	13-101
diskChange	Dos	52	13-15
diskChange	Workbench	25	13-101
diskChanged	TrackDisk	57	13-98
diskFont	Graphics	46	13-42
DiskFontHeader	DiskFont	39	13-10
diskFull	Dos	1	13-14
diskFull	FileSystem		12-28
diskInfo	Dos	44	13-15
diskinserted	InputEvent	17	13-57

diskInserted ..... Intuition 35 13-63
diskMagic ..... Workbench 16 13-101
diskName ..... Resources 7 13-93
diskNotValidated ..... Dos 50 13-13
DiskObject ..... Workbench 48 13-101
DiskObjectPtr ..... Workbench 26 13-101
diskremoved ..... InputEvent 17 13-57
diskRemoved ..... Intuition 36 13-63
disksync ..... Hardware 42 13-50
diskType ..... Dos 51 13-15
diskVersion ..... Workbench 17 13-101
diskWriteProtected ..... Dos 51 13-13
diskWriteProtected ..... FileSystem 12-28
DisownBlitter ..... Graphics 9 13-47
Dispatch ..... Exec 53 13-25
DisplayAlert ..... Intuition 16 13-73
DisplayBeep ..... Intuition 19 13-73
DisplayFlags ..... Graphics 41 13-40
DisplayFlagSet ..... Graphics 42 13-40
DisplayMode ..... Intuition 3 13-69
DisposeLayerInfo ..... Layers 34 13-80
DisposeRegion ..... Graphics 10 13-47
Divs32 ..... Arts 12- 9
divsW ..... Assembler 12-13
Divu32 ..... Arts 12- 9
divuW ..... Assembler 12-13
dle ..... ASCII 12-11
dm0 ..... Graphics 41 13-42
dmaAll ..... Hardware 37 13-50
DmaFlags ..... Hardware 30 13-50
DmaFlagSet ..... Hardware 34 13-50
dmaSet ..... Hardware 33 13-50
dModeCount ..... Intuition 6 13-69
DoCollision ..... Graphics 11 13-47
DoIO ..... Exec 54 13-25
done ..... FFPInOut 12-24
done ..... FileSystem 12-28
done ..... InOut 12-36
done ..... LongRealInOut 12-41
done ..... RealInOut 12-50
dosCmdBuf ..... Arts 12- 8
dosCmdLen ..... Arts 12- 8
dosDisk ..... Dos 30 13-13
DosInfo ..... Dos 26 13-16
DosInfoPtr ..... Dos 16 13-16
DosLibrary ..... Dos 1 13-16
DosLibraryPtr ..... Dos 9 13-16
dosName ..... Dos 13 13-12
DosPacket ..... Dos 2 13-15
DosPacketPtr ..... Dos 16 13-15
DoubleClick ..... Intuition 20 13-73
downBackGadget ..... Intuition 17 13-69
downKeys ..... GamePort 22 13-34
downup ..... KeyMap 14 13-78
dp2dFacShift ..... KeyMap 21 13-78
dp2dIndexMask ..... KeyMap 20 13-78
dpb1 ..... KeyMap 16 13-78
dpb2 ..... KeyMap 16 13-78
dpbDead ..... KeyMap 16 13-78
dpbMod ..... KeyMap 16 13-78
draft ..... Intuition 20 13-68
dragGadget ..... Intuition 18 13-69
Draw ..... Graphics 12 13-47
DrawBorder ..... Intuition 25 13-73
DrawCircle ..... GfxMacros 32 13-35
DrawEllipse ..... Graphics 15 13-47
drawer ..... Workbench 23 13-101
DrawerData ..... Workbench 4 13-102
drawerDataFileSize ..... Workbench 11 13-102
DrawerDataPtr ..... Workbench 27 13-101
DrawGList ..... Graphics 20 13-47
DrawImage ..... Intuition 29 13-73
DrawModes ..... Graphics 41 13-42
DrawModeSet ..... Graphics 42 13-42

drf4 ..... Resources 45 13-92
drf5 ..... Resources 45 13-92
drf6 ..... Resources 45 13-92
drive35 ..... TrackDisk 46 13-98
drive525 ..... TrackDisk 47 13-98
driveInUse ..... TrackDisk 5 13-99
drt37422D2S ..... Resources 10 13-93
dskblk ..... Hardware 41 13-50
dskChange ..... Hardware 49 13-53
dskDirec ..... Hardware 3 13-54
dskDmaOff ..... Resources 8 13-93
dskMotor ..... Hardware 5 13-54
dskProt ..... Hardware 49 13-53
dskRdy ..... Hardware 49 13-53
dskSel0 ..... Hardware 3 13-54
dskSel1 ..... Hardware 3 13-54
dskSel2 ..... Hardware 3 13-54
dskSel3 ..... Hardware 3 13-54
dskSide ..... Hardware 3 13-54
dskStep ..... Hardware 3 13-54
dskTrack0 ..... Hardware 49 13-53
dsrCpr ..... Console 3 13- 7
dualpf ..... Graphics 30 13-44
dumpRPort ..... Printer 19 13-87
DupLock ..... Dos 38 13-17
e MathLib0 12-42
e ..... MathLibFFP 12-44
e ..... MathLibLong 12-46
e0 ..... Printer 52 13-88
e0 ..... Serial 42 13-95
eightLPI ..... Intuition 23 13-68
elite ..... Intuition 18 13-68
em ..... ASCII 12-11
empty ..... Resources 11 13-93
Enable ..... Exec 55 13-25
end ..... Dos 25 13-12
endGadget ..... Intuition 5 13-61
EndRefresh ..... Intuition 33 13-73
EndRequest ..... Intuition 35 13-73
EndUpdate ..... Layers 35 13-80
enq ..... ASCII 12-11
Enqueue ..... Exec 56 13-25
eof ..... ASCII 12-11
eofMode ..... Parallel 25 13-86
eofMode ..... Serial 32 13-95
eol ..... ASCII 12-11
eol ..... InOut 12-36
eorB ..... Assembler 12-13
eoriB ..... Assembler 12-13
eoriCCR ..... Assembler 12-13
eoriL ..... Assembler 12-13
eoriSR ..... Assembler 12-13
eoriW ..... Assembler 12-13
eorL ..... Assembler 12-13
eorW ..... Assembler 12-13
eot ..... ASCII 12-11
epson ..... Intuition 13 13-67
epsonJX80 ..... Intuition 13 13-67
Error ..... Arts 12- 6
error ..... Dos 14 13-14
Error ..... Printer 51 13-88
Error ..... Serial 41 13-95
ErrorFrame ..... Arts 12- 7
errorFrame ..... Arts 12- 8
ErrorType ..... Arts 12- 7
errSetCType ..... GamePort 19 13-34
esc ..... ASCII 12-11
etb ..... ASCII 12-11
etx ..... ASCII 12-11
event ..... Dos 28 13-15
event ..... InputEvent 15 13-57
eventMax ..... Intuition 7 13-69
Examine ..... Dos 39 13-17
examineNext ..... Dos 43 13-15

examineObject ..... Dos 42 13-15
except ..... Exec 9 13-21
exception ..... Arts 12- 7
Exception ..... Exec 1 13-26
exception ..... Exec 6 13-21
exclusiveLock ..... Dos 31 13-12
execBase ..... Exec 53 13-24
ExecBase ..... Exec 55 13-23
ExecBasePtr ..... Exec 50 13-24
Execute ..... Dos 42 13-17
execute ..... Dos 48 13-12
exgAA ..... Assembler 12-13
exgDA ..... Assembler 12-13
exgDD ..... Assembler 12-13
Exit ..... Dos 46 13-17
ExitIntr ..... Exec 2 13-26
ExNext ..... Dos 47 13-17
Exp ..... MathIEEEDoubTrans 13 13-82
exp MathLib0 12-42
exp ..... MathLibFFP 12-44
exp ..... MathLibLong 12-46
Exp ..... MathTrans 14 13-83
expansionBase ..... Expansion 49 13-30
ExpansionControl ..... Expansion 26 13-30
ExpansionRom ..... Expansion 13 13-30
expansionSize ..... Expansion 50 13-30
expansionSlots ..... Expansion 51 13-30
expansionUnused ..... Expansion 28 13-32
explicit ..... Arts 12- 7
expunge ..... Exec 48 13-21
expunged ..... Narrator 22 13-84
extClear ..... TrackDisk 36 13-98
extCom ..... TrackDisk 29 13-98
extend ..... Printer 48 13-88
extended ..... Graphics 43 13-42
exter ..... Hardware 42 13-50
extFormat ..... TrackDisk 34 13-98
extFunc ..... Exec 47 13-21
extL ..... Assembler 12-13
extMotor ..... TrackDisk 32 13-98
extraHalfbrite ..... Graphics 29 13-44
extRawRead ..... TrackDisk 37 13-98
extRawWrite ..... TrackDisk 38 13-98
extRead ..... TrackDisk 31 13-98
extSeek ..... TrackDisk 33 13-98
extUpdate ..... TrackDisk 35 13-98
extW ..... Assembler 12-13
extWrite ..... TrackDisk 30 13-98
fail ..... Dos 15 13-14
fanfold ..... Intuition 36 13-68
fast ..... Exec 6 13-20
fast ..... Hardware 18 13-50
FatIntuiMessage ..... Intuition 33 13-69
FattenLayerInfo ..... Layers 36 13-80
fchId ..... DiskFont 18 13-10
fctReturn ..... Arts 12- 7
female ..... Narrator 33 13-84
ff ..... ASCII 12-11
Fieee ..... MathIEEEDoubTrans 14 13-82
Fieee ..... MathTrans 15 13-83
File ..... FileSystem 12-28
FileHandle ..... Dos 46 13-14
FileHandlePtr ..... Dos 39 13-12
FileInfoBlock ..... Dos 54 13-12
FileInfoBlockPtr ..... Dos 53 13-12
FileLock ..... Dos 17 13-17
FileLockPtr ..... Dos 40 13-12
FileMode ..... FileSystem 12-28
FileModeSet ..... FileSystem 12-28
filenameSize ..... Intuition 4 13-67
fileNotObject ..... Dos 37 13-13
fillCarryIn ..... Hardware 11 13-51
fillOr ..... Hardware 11 13-51
fillXor ..... Hardware 11 13-51

FindConfigDev ..... Expansion 29 13-32
FindName ..... Exec 3 13-26
FindPort ..... Exec 5 13-26
FindResident ..... Exec 6 13-26
FindSemaphore ..... Exec 8 13-26
FindTask ..... Exec 10 13-26
FindToolType ..... Icon 18 13-55
fine ..... Intuition 19 13-68
fineScroll ..... Graphics 48 13-38
fineScrollMask ..... Graphics 50 13-38
fineScrollShift ..... Graphics 49 13-38
finish ..... Audio 20 13- 3
first ..... Str 12-53
first ..... Strings 12-56
firstDot ..... Graphics 51 13-42
FirstPos ..... Str 12-53
flg ..... Hardware 33 13-53
Flood ..... Graphics 22 13-47
flush ..... Dos 46 13-15
flush ..... Exec 35 13-22
fnt0 ..... Printer 3 13-88
fnt1 ..... Printer 4 13-88
fnt10 ..... Printer 13 13-88
fnt2 ..... Printer 5 13-88
fnt3 ..... Printer 6 13-88
fnt4 ..... Printer 7 13-88
fnt5 ..... Printer 8 13-88
fnt6 ..... Printer 9 13-88
fnt7 ..... Printer 10 13-88
fnt8 ..... Printer 11 13-88
fnt9 ..... Printer 12 13-88
followMouse ..... Intuition 5 13-61
font ..... Exec 17 13-19
FontContents ..... DiskFont 27 13-10
FontContentsHeader ..... DiskFont 33 13-10
FontContentsHeaderPtr ..... DiskFont 38 13-10
FontFlags ..... Graphics 45 13-42
FontFlagSet ..... Graphics 49 13-42
FontStyles ..... Graphics 43 13-42
FontStyleSet ..... Graphics 44 13-42
Forbid ..... Exec 11 13-26
format ..... TrackDisk 17 13-98
fourColor ..... PrtBase 16 13-91
fracCols ..... Printer 13 13-89
fracRows ..... Printer 13 13-89
free ..... Audio 18 13- 3
FreeBoardMem ..... Expansion 32 13-32
FreeColorMap ..... Graphics 26 13-47
FreeConfigDev ..... Expansion 34 13-32
FreeCopList ..... Graphics 27 13-47
FreeCprList ..... Graphics 28 13-47
FreeDiskObject ..... Icon 40 13-55
FreeEntry ..... Exec 12 13-26
FreeExpansionMem ..... Expansion 35 13-32
FreeFreeList ..... Icon 16 13-55
FreeGBuffers ..... Graphics 29 13-47
freeHoriz ..... Intuition 4 13-62
FreeList ..... Workbench 44 13-101
FreeListPtr ..... Workbench 28 13-101
freeLock ..... Dos 34 13-15
FreeMem ..... Exec 13 13-26
FreeMiscResource ..... Resources 43 13-93
freeMsg ..... Exec 16 13-19
FreePotBits ..... Resources 2 13-94
FreeRaster ..... Graphics 32 13-47
FreeRemember ..... Intuition 37 13-73
FreeSignal ..... Exec 15 13-26
FreeSprite ..... Graphics 35 13-47
FreeSysRequest ..... Intuition 39 13-73
FreeTrap ..... Exec 16 13-26
FreeUnit ..... Resources 15 13-93
freeVert ..... Intuition 4 13-62
FreeVPortCopLists ..... Graphics 36 13-47
FreeWBObject ..... Icon 28 13-55

freqErr	Narrator	28	13-84
fs	ASCII		12-11
fullCols	Printer	13	13-89
fullRows	Printer	13	13-89
gadgDisabled	Intuition	2	13-61
Gadget	Intuition	33	13-61
gadget0002	Intuition	15	13-61
gadgetBackFill	Workbench	18	13-101
gadgetCount	Intuition	22	13-69
gadgetdown	InputEvent	15	13-57
gadgetDown	Intuition	34	13-63
GadgetFlags	Intuition	56	13-60
GadgetFlagSet	Intuition	3	13-61
GadgetInfo	Intuition	6	13-70
GadgetPtr	Intuition	27	13-59
Gadgets	Intuition	16	13-69
gadgetsLock	Intuition	26	13-69
gadgetType	Intuition	30	13-61
gadgetup	InputEvent	16	13-57
gadgetUp	Intuition	34	13-63
gadgHBox	Intuition	57	13-60
gadgHighbits	Intuition	12	13-61
gadgHImage	Intuition	57	13-60
gadgHNone	Intuition	13	13-61
gadgImage	Intuition	57	13-60
gadgImmediate	Intuition	5	13-61
gamePort0	Hardware	49	13-53
gamePort1	Hardware	51	13-53
gamePortName	GamePort	12	13-34
GamePortTrigger	GamePort	24	13-34
garbage	Workbench	23	13-101
gelGone	Graphics	10	13-39
GelsInfo	Graphics	26	13-42
GelsInfoPtr	Graphics	29	13-36
genloc	Graphics	41	13-40
genlocAudio	Graphics	30	13-44
genlocVideo	Graphics	29	13-44
GetArg	Arguments		12- 3
getBlock	Dos	25	13-15
GetCC	Exec	17	13-26
GetColorMap	Graphics	37	13-47
GetCurrentBinding	Expansion	37	13-32
GetDefPrefs	Intuition	40	13-73
GetDiskObject	Icon	35	13-55
getDriveType	TrackDisk	24	13-98
GetExtension	FileNames		12-26
GetGBuffers	Graphics	39	13-47
GetIcon	Icon	30	13-55
GetLock	Arguments		12- 3
GetMsg	Exec	18	13-26
getNumTracks	TrackDisk	25	13-98
GetPacket	Dos	52	13-17
GetPath	FileNames		12-26
GetPos	FileSystem		12-29
GetPos	Windows		12-64
GetPrefs	Intuition	42	13-73
GetRGB4	Graphics	42	13-47
GetScreenData	Intuition	44	13-73
GetSeed	RandomNumber		12-47
GetSize	Windows		12-64
GetSprite	Graphics	44	13-47
getSysTime	Timer	16	13-97
GetUnit	Resources	17	13-93
GetUnitID	Resources	21	13-93
GetWBObject	Icon	25	13-55
gfx	PrtBase	5	13-91
GfxBase	Graphics	43	13-40
GfxBasePtr	Graphics	29	13-41
gimmeZeroZero	Intuition	36	13-64
GiveUnit	Resources	24	13-93
GListEnv	Intuition	51	13-69
GListEnvPtr	Intuition	5	13-70
GraphicsReserved1	Graphics	46	13-47
GraphicsReserved2	Graphics	47	13-47

green ..... Console 28 13- 6
greenBg ..... Console 37 13- 6
gRelBottom ..... Intuition 57 13-60
gRelHeight ..... Intuition 1 13-61
gRelRight ..... Intuition 57 13-60
gRelWidth ..... Intuition 1 13-61
gs ..... ASCII 12-11
gzzGadget ..... Intuition 27 13-61
halt ..... Arts 12- 7
ham ..... Graphics 30 13-44
hardChannels ..... Audio 15 13- 3
hidden ..... Dos 48 13-12
highBox ..... Intuition 5 13-60
highComp ..... Intuition 5 13-60
highItem ..... Intuition 6 13-60
highNone ..... Intuition 11 13-60
hires ..... Graphics 31 13-44
hiresGadget ..... Intuition 10 13-69
hiresPick ..... Intuition 3 13-69
holdnmodify ..... Graphics 43 13-38
horizPos ..... Graphics 53 13-38
hpLaserjet ..... Intuition 14 13-67
hpLaserjetPlus ..... Intuition 15 13-67
hSizeBits ..... Hardware 47 13-50
hSizeMask ..... Hardware 49 13-50
ht ..... ASCII 12-11
hts ..... Printer 40 13-88
iBaseLock ..... Intuition 27 13-69
IBox ..... Intuition 37 13-69
IDCMPFlags ..... Intuition 32 13-63
IDCMPFlagSet ..... Intuition 40 13-63
idDos ..... BootBlock 19 13- 4
idKick ..... BootBlock 20 13- 4
if5 ..... Hardware 33 13-53
if6 ..... Hardware 33 13-53
ignore ..... Exec 38 13-20
illCall ..... Arts 12- 7
illCase ..... Arts 12- 7
illegal ..... Assembler 12-14
ILocks ..... Intuition 25 13-69
Image ..... Intuition 20 13-63
imageNegative ..... Intuition 25 13-68
imagePositive ..... Intuition 24 13-68
ImagePtr ..... Intuition 28 13-59
imm ..... Assembler 12-12
inactivewindow ..... InputEvent 19 13-57
inactiveWindow ..... Intuition 36 13-63
ind ..... Printer 23 13-87
indexSync ..... TrackDisk 44 13-98
info ..... Dos 45 13-15
Info ..... Dos 53 13-17
Info ..... KeyMap 38 13-78
InfoData ..... Dos 10 13-13
InfoDataPtr ..... Dos 9 13-13
InfoPtr ..... KeyMap 39 13-78
inhibit ..... Dos 50 13-15
InitAnimate ..... GfxMacros 13 13-35
InitArea ..... Graphics 48 13-47
InitBitMap ..... Graphics 51 13-47
InitCode ..... Exec 19 13-26
initErr ..... Parallel 24 13-86
initErr ..... Serial 43 13-95
InitGels ..... Graphics 55 13-47
InitGMasks ..... Graphics 1 13-48
InitLayers ..... Layers 38 13-80
InitMasks ..... Graphics 2 13-48
InitRastPort ..... Graphics 3 13-48
InitRequester ..... Intuition 49 13-73
InitResident ..... Exec 21 13-26
InitSemaphore ..... Exec 23 13-26
InitStruct ..... Exec 25 13-26
InitTmpRas ..... Graphics 4 13-48
InitView ..... Graphics 7 13-48
InitVPort ..... Graphics 8 13-48

Input	Dos	56	13-17
InputEvent	InputEvent	53	13-57
InputEventPtr	InputEvent	52	13-57
inputName	Input	12	13-56
inRequest	Intuition	36	13-64
Insert	Exec	28	13-26
Insert	Strings		12-56
InstallClipRegion	Layers	40	13-80
int2pend	Expansion	13	13-31
int6pend	Expansion	14	13-31
int7pend	Expansion	15	13-31
inTask	Exec	15	13-22
inten	Hardware	42	13-50
intena	Expansion	11	13-31
interlace	Graphics	39	13-38
internalMemory	Printer	53	13-88
interrupt	Exec	16	13-19
Interrupt	Exec	47	13-19
interrupt	InputEvent	48	13-57
interrupting	Expansion	16	13-31
InterruptPtr	Exec	52	13-19
IntFlags	Hardware	40	13-50
IntFlagSet	Hardware	44	13-50
intSet	Hardware	43	13-50
IntuiMessage	Intuition	18	13-64
IntuiMessagePtr	Intuition	29	13-59
IntuiText	Intuition	55	13-62
IntuiTextLength	Intuition	50	13-73
IntuiTextPtr	Intuition	30	13-59
intuiTicks	Intuition	37	13-63
Intuition	Intuition	52	13-73
IntuitionBase	Intuition	19	13-70
IntuitionBasePtr	Intuition	21	13-72
IntVector	Exec	53	13-19
inUse	FileSystem		12-28
inval	Exec	9	13-21
invalid	Exec	27	13-22
invalidComponentName	Dos	47	13-13
invalidLock	Dos	48	13-13
invalidResidentLibrary	Dos	38	13-13
invBaud	Serial	42	13-95
inversvid	Graphics	41	13-42
invertHam	Printer	52	13-88
invMd	Windows		12-64
invParam	Parallel	22	13-86
invParam	Serial	42	13-95
IOAudio	Audio	35	13- 3
IOAudioPtr	Audio	45	13- 3
IOClipboard	Clipboard	26	13- 5
IOClipboardPtr	Clipboard	39	13- 5
IODRPReq	Printer	23	13-89
IODRPReqPtr	Printer	42	13-89
IoErr	Dos	50	13-17
IOFlagSet	Exec	42	13-22
IONarrator	Narrator	52	13-84
IONarratorPtr	Narrator	10	13-85
IOParallel	Parallel	29	13-86
IOParallelPtr	Parallel	36	13-86
IOPArray	Parallel	17	13-86
IOPrinter	Printer	55	13-88
IOPrinterPtr	Printer	11	13-89
ioProc	Workbench	25	13-101
IORequest	Exec	48	13-22
IORequestPtr	Exec	56	13-22
IOSerial	Serial	46	13-95
IOSerialPtr	Serial	3	13-96
IOStdReq	Exec	57	13-22
IOStdReqPtr	Exec	12	13-23
IOTArray	Serial	26	13-95
IOTimer	Timer	27	13-97
IOTimerPtr	Timer	31	13-97
IOTrackDisk	TrackDisk	9	13-99
IOTrackDiskPtr	TrackDisk	14	13-99
ir	Hardware	36	13-53

isDrawn ..... Intuition 5 13-60
isGrtrX ..... Graphics 39 13-37
isGrtrY ..... Graphics 40 13-37
IsInteractive ..... Dos 57 13-17
isLessX ..... Graphics 37 13-37
isLessY ..... Graphics 38 13-37
Isrvstr ..... Graphics 35 13-41
IsrvstrPtr ..... Graphics 30 13-36
iStateLock ..... Intuition 26 13-69
italic ..... Console 23 13- 6
italic ..... Graphics 43 13-42
ItemAddress ..... Intuition 53 13-73
itemEnabled ..... Intuition 4 13-60
itemText ..... Intuition 4 13-60
jam1 ..... Graphics 34 13-43
jam2 ..... Graphics 35 13-43
jfy0 ..... Printer 22 13-88
jfy1 ..... Printer 24 13-88
jfy3 ..... Printer 23 13-88
jfy5 ..... Printer 19 13-88
jfy6 ..... Printer 21 13-88
jfy7 ..... Printer 20 13-88
jmp ..... Assembler 12-14
jsr ..... Assembler 12-14
k2 ..... GamePort 22 13-34
k3 ..... GamePort 22 13-34
k4 ..... GamePort 22 13-34
k5 ..... GamePort 22 13-34
k6 ..... GamePort 22 13-34
k7 ..... GamePort 22 13-34
k8 ..... GamePort 22 13-34
keyboardName ..... Keyboard 12 13-77
keyCodeB ..... Intuition 52 13-63
keyCodeFirst ..... InputEvent 24 13-57
keyCodeLast ..... InputEvent 25 13-57
keyCodeM ..... Intuition 54 13-63
keyCodeN ..... Intuition 53 13-63
keyCodeQ ..... Intuition 49 13-63
keyCodeV ..... Intuition 51 13-63
keyCodeX ..... Intuition 50 13-63
KeyInfo ..... KeyMap 30 13-78
KeyMap ..... KeyMap 40 13-78
KeyMapNode ..... KeyMap 51 13-78
KeyMapPtr ..... KeyMap 50 13-78
KeyMapResource ..... KeyMap 55 13-78
KeyMapTypes ..... KeyMap 14 13-78
KeyMapTypeSet ..... KeyMap 15 13-78
Keys ..... GamePort 22 13-34
KeySet ..... GamePort 23 13-34
kick ..... Workbench 24 13-101
kickstartDisk ..... Dos 32 13-13
kmp4 ..... KeyMap 14 13-78
knobHit ..... Intuition 6 13-62
knobHmin ..... Intuition 11 13-62
knobVmin ..... Intuition 10 13-62
labelSize ..... TrackDisk 43 13-98
lace ..... Graphics 29 13-44
lAlt ..... InputEvent 47 13-57
largest ..... Exec 8 13-20
Largest ..... Heap 12-33
last ..... Str 12-53
last ..... Strings 12-56
lastComm ..... TrackDisk 28 13-98
LastPos ..... Str 12-53
latin1First ..... InputEvent 35 13-57
latin1Last ..... InputEvent 36 13-57
launch ..... Exec 7 13-21
Layer ..... Graphics 53 13-36
layerBackdrop ..... Graphics 44 13-41
layerClipRectsLost ..... Graphics 45 13-41
LayerFlags ..... Graphics 42 13-41
LayerFlagSet ..... Graphics 46 13-41
LayerInfo ..... Graphics 47 13-41
layerInfoLock ..... Intuition 26 13-69

LayerInfoPtr	Graphics	32	13-36
LayerPtr	Graphics	31	13-36
layerRefresh	Graphics	44	13-41
layerRomLock	Intuition	26	13-69
layerSimple	Graphics	43	13-41
layerSmart	Graphics	43	13-41
layerSuper	Graphics	43	13-41
layerUpdating	Graphics	43	13-41
lButton	InputEvent	37	13-57
lCommand	InputEvent	47	13-57
lea	Assembler		12-14
led	Hardware	49	13-53
leftBorder	Intuition	6	13-61
leftButton	InputEvent	49	13-57
leftHit	Graphics	44	13-37
Length	FileSystem		12-29
Length	Str		12-53
Length	Strings		12-56
letter	Intuition	21	13-68
lf	ASCII		12-11
lf3	Graphics	43	13-41
lf5	Graphics	43	13-41
lib	Arts		12- 7
LibFlags	Exec	53	13-21
LibFlagSet	Exec	54	13-21
library	Exec	17	13-19
Library	Exec	55	13-21
LibraryPtr	Exec	10	13-22
lineErr	Parallel	22	13-86
lineErr	Serial	42	13-95
lineMode	Hardware	11	13-51
lineTooLong	Dos	36	13-13
linkW	Assembler		12-14
List	Exec	33	13-19
ListPtr	Exec	40	13-19
lmnRegion	Graphics	7	13-42
lms	Printer	32	13-88
ln	MathLib0		12-42
ln	MathLibFFP		12-44
ln	MathLibLong		12-46
LoadRGB4	Graphics	9	13-48
LoadSeg	Dos	2	13-18
LoadView	Graphics	12	13-48
locateObject	Dos	30	13-15
lock	Audio	22	13- 3
Lock	Dos	3	13-18
lockErr	FileSystem		12-28
LockIBase	Intuition	56	13-73
LockLayer	Layers	43	13-80
LockLayerInfo	Layers	44	13-80
LockLayerRom	Graphics	13	13-48
LockLayers	Layers	45	13-80
lof	Graphics	52	13-37
Log	MathIEEEDoubTrans	15	13-82
Log	MathTrans	16	13-83
Log10	MathIEEEDoubTrans	16	13-82
Log10	MathTrans	17	13-83
lonelyMessage	Intuition	39	13-63
longint	Intuition	7	13-61
Lookup	FileSystem		12-28
lowCheckWidth	Intuition	14	13-60
lowCommWidth	Intuition	15	13-60
lowresGadget	Intuition	10	13-69
lowresPick	Intuition	3	13-69
ls0	Assembler		12-12
ls1	Assembler		12-12
ls10	Assembler		12-12
ls11	Assembler		12-12
ls12	Assembler		12-12
ls13	Assembler		12-12
ls14	Assembler		12-12
ls15	Assembler		12-12
ls2	Assembler		12-12
ls3	Assembler		12-12

ls4 ..... Assembler 12-12
ls5 ..... Assembler 12-12
ls6 ..... Assembler 12-12
ls7 ..... Assembler 12-12
ls8 ..... Assembler 12-12
ls9 ..... Assembler 12-12
lShift ..... InputEvent 47 13-57
lslB ..... Assembler 12-14
lsliB ..... Assembler 12-14
lsliL ..... Assembler 12-14
lsliW ..... Assembler 12-14
lslL ..... Assembler 12-14
lslmW ..... Assembler 12-14
lslW ..... Assembler 12-14
lsrB ..... Assembler 12-14
lsriB ..... Assembler 12-14
lsriL ..... Assembler 12-14
lsriW ..... Assembler 12-14
lsrL ..... Assembler 12-14
lsrmW ..... Assembler 12-14
lsrW ..... Assembler 12-14
m640 ..... Graphics 44 13-38
m68010 ..... Exec 51 13-23
m68020 ..... Exec 51 13-23
m68881 ..... Exec 51 13-23
magenta ..... Console 31 13- 6
magentaBg ..... Console 40 13- 6
makeBad ..... Narrator 17 13-84
makeBad ..... Translator 13 13-100
MakeDosNode ..... Expansion 40 13-32
MakeFileName ..... FileNames 12-26
MakeFunctions ..... Exec 31 13-26
MakeLibrary ..... Exec 34 13-26
MakeScreen ..... Intuition 1 13-74
MakeVPort ..... Graphics 14 13-48
male ..... Narrator 32 13-84
mark ..... Serial 18 13-95
mAsm ..... Console 10 13- 7
master ..... Hardware 32 13-50
MatchToolValue ..... Icon 21 13-55
matchword ..... Exec 47 13-23
mAwm ..... Console 11 13- 7
maxBody ..... Intuition 12 13-62
maxBytesPerRow ..... Hardware 51 13-50
maxFontName ..... DiskFont 17 13-10
maxFontPath ..... DiskFont 16 13-10
maxFreq ..... Narrator 47 13-84
maxKeys ..... Arts 12- 7
maxKeys ..... KeyMap 22 13-78
maxModName ..... Arts 12- 7
maxPitch ..... Narrator 45 13-84
maxPot ..... Intuition 13 13-62
maxRate ..... Narrator 43 13-84
maxTabs ..... ConUnit 26 13- 8
maxVol ..... Narrator 49 13-84
mButton ..... InputEvent 39 13-57
memBit ..... Expansion 3 13-31
MemChunk ..... Exec 12 13-20
MemChunkPtr ..... Exec 11 13-20
memClear ..... Exec 7 13-20
MemEntry ..... Exec 25 13-20
memErr ..... FileSystem 12-28
MemHeader ..... Exec 16 13-20
MemHeaderPtr ..... Exec 24 13-20
MemList ..... Exec 32 13-20
memList ..... Expansion 7 13-31
MemListPtr ..... Exec 37 13-20
memMask ..... Expansion 2 13-31
memory ..... DiskFont 23 13-10
memory ..... Exec 17 13-19
memoryBase ..... Expansion 52 13-30
memorySize ..... Expansion 53 13-30
memorySlots ..... Expansion 54 13-30
MemReqs ..... Exec 5 13-20

MemReqSet ..... Exec 9 13-20
memSize ..... Expansion 4 13-31
memSpace ..... Expansion 8 13-31
MemTypeSet ..... Exec 10 13-20
Menu ..... Intuition 46 13-59
menuCancel ..... Intuition 3 13-64
menuDown ..... Intuition 46 13-63
menuEnabled ..... Intuition 42 13-59
menuHot ..... Intuition 2 13-64
MenuItem ..... Intuition 18 13-60
MenuItemFlags ..... Intuition 3 13-60
MenuItemFlagSet ..... Intuition 8 13-60
MenuItemPtr ..... Intuition 31 13-59
menulist ..... InputEvent 16 13-57
menuNull ..... Intuition 47 13-63
menuPick ..... Intuition 34 13-63
MenuPtr ..... Intuition 32 13-59
menuState ..... Intuition 37 13-64
menuToggle ..... Intuition 4 13-60
menuToggled ..... Intuition 7 13-60
menuUp ..... Intuition 45 13-63
menuVerify ..... Intuition 35 13-63
menuWaiting ..... Intuition 4 13-64
message ..... Exec 16 13-19
Message ..... Exec 56 13-20
MessagePtr ..... Exec 4 13-21
mfmPrec ..... Hardware 18 13-50
microHz ..... Timer 18 13-97
midButton ..... InputEvent 48 13-57
miDrawn ..... Intuition 43 13-59
mif10 ..... Intuition 5 13-60
mif11 ..... Intuition 5 13-60
mif5 ..... Intuition 4 13-60
mif9 ..... Intuition 5 13-60
milCols ..... Printer 13 13-89
milRows ..... Printer 13 13-89
minFreq ..... Narrator 46 13-84
MinList ..... Exec 41 13-19
MinListPtr ..... Exec 46 13-19
MinNode ..... Exec 29 13-19
MinNodePtr ..... Exec 28 13-19
minPitch ..... Narrator 44 13-84
minRate ..... Narrator 42 13-84
minVol ..... Narrator 48 13-84
miscName ..... Resources 27 13-93
MiscResource ..... Resources 33 13-93
MiscResourcePtr ..... Resources 37 13-93
mLnm ..... Console 9 13- 7
mod ..... Arts 12- 7
Mode ..... Windows 12-64
modeErr ..... Narrator 27 13-84
ModeSet ..... Windows 12-64
ModifyIDCMP ..... Intuition 2 13-74
ModifyProp ..... Intuition 4 13-74
ModKeys ..... Arts 12- 7
ModName ..... Arts 12- 7
ModType ..... Arts 12- 7
ModuleId ..... Arts 12- 7
moduleInfo ..... Arts 12- 8
ModuleInfo ..... Arts 12- 8
ModuleInfoPtr ..... Arts 12- 8
moreCache ..... Dos 37 13-15
motor ..... TrackDisk 15 13-98
mouse ..... GamePort 30 13-34
mouseButtons ..... Intuition 33 13-63
mouseMove ..... Intuition 33 13-63
Mouth ..... Narrator 11 13-85
MouthPtr ..... Narrator 18 13-85
Move ..... Graphics 16 13-48
move ..... Graphics 48 13-37
moveaL ..... Assembler 12-14
moveaW ..... Assembler 12-14
moveB ..... Assembler 12-14
moveCCRfr ..... Assembler 12-14

moveCCRto ..... Assembler 12-14
movecLfr ..... Assembler 12-14
movecLto ..... Assembler 12-14
moveL ..... Assembler 12-14
MoveLayer ..... Layers 46 13-80
MoveLayerInFrontOf ..... Layers 48 13-80
movemL ..... Assembler 12-14
movemmL ..... Assembler 12-14
movemmW ..... Assembler 12-14
movemW ..... Assembler 12-14
movepL ..... Assembler 12-14
movepmL ..... Assembler 12-14
movepmW ..... Assembler 12-14
movepW ..... Assembler 12-14
moveqL ..... Assembler 12-14
movesB ..... Assembler 12-14
MoveScreen ..... Intuition 12 13-74
movesL ..... Assembler 12-14
MoveSprite ..... Graphics 19 13-48
moveSRfr ..... Assembler 12-14
moveSRto ..... Assembler 12-14
movesW ..... Assembler 12-14
moveUSPfr ..... Assembler 12-14
moveUSPto ..... Assembler 12-14
moveW ..... Assembler 12-14
MoveWindow ..... Intuition 15 13-74
moving ..... Windows 12-64
mr10 ..... Exec 7 13-20
mr11 ..... Exec 7 13-20
mr12 ..... Exec 7 13-20
mr13 ..... Exec 7 13-20
mr14 ..... Exec 7 13-20
mr15 ..... Exec 7 13-20
mr3 ..... Exec 6 13-20
mr4 ..... Exec 6 13-20
mr5 ..... Exec 6 13-20
mr6 ..... Exec 6 13-20
mr7 ..... Exec 6 13-20
mr8 ..... Exec 7 13-20
mr9 ..... Exec 7 13-20
MrgCop ..... Graphics 23 13-48
msbSync ..... Hardware 18 13-50
msgPort ..... Exec 16 13-19
MsgPort ..... Exec 40 13-20
MsgPortAction ..... Exec 38 13-20
MsgPortPtr ..... Exec 55 13-20
mSpOn ..... Serial 19 13-95
mt0 ..... Workbench 25 13-101
MType ..... Workbench 25 13-101
Muls32 ..... Arts 12- 9
mulsW ..... Assembler 12-14
multiBroadcast ..... InputEvent 48 13-57
Mulu32 ..... Arts 12- 9
muluW ..... Assembler 12-14
mustDraw ..... Graphics 9 13-39
nabc ..... Hardware 55 13-50
nabnc ..... Hardware 55 13-50
nak ..... ASCII 12-11
nameDos ..... BootBlock 21 13- 4
nameKick ..... BootBlock 22 13- 4
nanbc ..... Hardware 55 13-50
nanbnc ..... Hardware 55 13-50
narratorName ..... Narrator 14 13-84
natural ..... Narrator 34 13-84
nbcdB ..... Assembler 12-14
nbit ..... Assembler 12-12
needsNoConcealedRasters ... Graphics 36 13-37
negative ..... Console 25 13- 6
negB ..... Assembler 12-14
negL ..... Assembler 12-14
negW ..... Assembler 12-14
negxB ..... Assembler 12-14
negxL ..... Assembler 12-14
negxW ..... Assembler 12-14

nel	Printer	24	13-87
never	Expansion	36	13-31
newActive	InputEvent	41	13-57
newBoard	Expansion	1	13-31
newFile	Dos	21	13-12
NewLayerInfo	Layers	50	13-80
newLayerInfoCalled	Graphics	8	13-42
NewList	ExecSupport	13	13-29
NewModifyProp	Intuition	18	13-74
newprefs	InputEvent	17	13-57
newPrefs	Intuition	35	13-63
NEWPROCESS	Coroutines		12-20
NewRegion	Graphics	24	13-48
NewScreen	Intuition	17	13-65
newSize	Intuition	33	13-63
NewWindow	Intuition	54	13-64
next	Graphics	50	13-37
nibbleWide	Expansion	32	13-31
nil	Dos	24	13-15
noAllocation	Audio	30	13- 3
noAudLib	Narrator	16	13-84
noBuffer	FileSystem		12-28
noButton	InputEvent	40	13-57
noCareRefresh	Intuition	37	13-64
noController	GamePort	30	13-34
noCrossFill	Graphics	51	13-42
nocts	Serial	44	13-95
Node	Exec	21	13-19
noDefaultDir	Dos	39	13-13
NodePtr	Exec	20	13-19
NodeType	Exec	15	13-19
noDisk	Dos	6	13-14
noDisk	FileSystem		12-28
noDiskPresent	Dos	28	13-13
nodsr	Serial	44	13-95
noEcho	FileNames		12-26
noFreeStore	Dos	34	13-13
noIconPosition	Workbench	19	13-101
noImp	Arts		12- 7
noisyReq	Intuition	33	13-60
noItem	Intuition	48	13-63
noMem	Narrator	15	13-84
noMem	TrackDisk	2	13-99
noMem	Translator	12	13-100
noMenu	Intuition	48	13-63
noMoreEntries	Dos	7	13-14
none	Arts		12- 7
nonstd	Exec	36	13-22
nonStd	Exec	46	13-21
noOccur	Str		12-53
nop	Assembler		12-14
nop	KeyMap	14	13-78
noQual	KeyMap	23	13-78
normalFont	Graphics	37	13-43
noSecHdr	TrackDisk	49	13-98
noShutup	Expansion	9	13-31
noSub	Intuition	48	13-63
notADosDisk	Dos	5	13-14
notB	Assembler		12-14
notdone	FileSystem		12-28
notDosDisk	FileSystem		12-28
notFound	FileSystem		12-28
notGraphics	Printer	52	13-88
notL	Assembler		12-14
notOpen	Parallel	22	13-86
notOpen	Serial	43	13-95
notReallyDos	Dos	31	13-13
notSpecified	TrackDisk	48	13-98
notUsed	Translator	11	13-100
notW	Assembler		12-14
noUnit	Audio	24	13- 3
noWait	Audio	28	13- 3
noWrite	Narrator	21	13-84
nTractor	Intuition	33	13-68

ntsc ..... Graphics 41 13-40
nul ..... ASCII 12-11
null ..... InputEvent 15 13-57
NumArgs ..... Arguments 12- 3
numericPad ..... InputEvent 48 13-57
numIEvents ..... Intuition 16 13-70
numILocks ..... Intuition 30 13-69
numSecs ..... TrackDisk 39 13-98
numUnits ..... TrackDisk 40 13-98
objectExists ..... Dos 41 13-13
objectInUse ..... Dos 40 13-13
objectNotFound ..... Dos 43 13-13
objectTooLarge ..... Dos 45 13-13
objectWrongType ..... Dos 49 13-13
obsoleteId ..... Clipboard 18 13- 5
ObtainConfigBinding ..... Expansion 42 13-32
ObtainSemaphore ..... Exec 40 13-26
ObtainSemaphoreList ..... Exec 42 13-26
Occurs ..... Strings 12-56
octant1 ..... Hardware 21 13-51
octant2 ..... Hardware 22 13-51
octant3 ..... Hardware 23 13-51
octant4 ..... Hardware 24 13-51
octant5 ..... Hardware 25 13-51
octant6 ..... Hardware 26 13-51
octant7 ..... Hardware 27 13-51
octant8 ..... Hardware 28 13-51
OffDisplay ..... GfxMacros 17 13-35
OffGadget ..... Intuition 27 13-74
OffMenu ..... Intuition 30 13-74
OffSprite ..... GfxMacros 19 13-35
OffVBlank ..... GfxMacros 21 13-35
ok ..... Dos 12 13-14
okAbort ..... Intuition 6 13-64
okCancel ..... Intuition 7 13-64
okimate20 ..... Intuition 13 13-67
okOk ..... Intuition 5 13-64
oldFile ..... Dos 20 13-12
OldOpenLibrary ..... Exec 43 13-26
OnDisplay ..... GfxMacros 16 13-35
oneDot ..... Graphics 51 13-42
oneDot ..... Hardware 16 13-51
OnGadget ..... Intuition 32 13-74
OnMenu ..... Intuition 35 13-74
OnSprite ..... GfxMacros 18 13-35
OnVBlank ..... GfxMacros 20 13-35
Open ..... Dos 12 13-18
open ..... Exec 50 13-21
OpenDevice ..... Exec 45 13-26
OpenDiskFont ..... DiskFont 4 13-11
openErr ..... FileSystem 12-28
OpenFont ..... Graphics 25 13-48
OpenInput ..... InOut 12-36
OpenIntuition ..... Intuition 37 13-74
OpenLibrary ..... Exec 49 13-26
OpenOutput ..... InOut 12-36
OpenResource ..... Exec 52 13-26
OpenScreen ..... Intuition 38 13-74
OpenWindow ..... Intuition 40 13-74
OpenWindow ..... Windows 12-64
OpenWorkBench ..... Intuition 42 13-74
optLib ..... Arts 12- 7
orB ..... Assembler 12-14
oriB ..... Assembler 12-14
oriCCR ..... Assembler 12-14
oriL ..... Assembler 12-14
oriSR ..... Assembler 12-14
oriW ..... Assembler 12-14
orL ..... Assembler 12-14
ormB ..... Assembler 12-14
ormL ..... Assembler 12-14
ormW ..... Assembler 12-14
OrRectRegion ..... Graphics 27 13-48
OrRegionRegion ..... Graphics 30 13-48

orW ..... Assembler 12-14
otherRefresh ..... Intuition 48 13-64
outlx ..... Resources 52 13-93
outly ..... Resources 52 13-93
Output ..... Dos 15 13-18
outrx ..... Resources 52 13-93
outry ..... Resources 52 13-93
outStep ..... Graphics 38 13-39
overlay ..... Graphics 9 13-39
overlay ..... Hardware 49 13-53
overrun ..... Serial 37 13-95
ovFlag ..... Hardware 11 13-51
OwnBlitter ..... Graphics 33 13-48
Pad ..... Hardware 10 13-54
pal ..... Graphics 41 13-40
paperOut ..... Parallel 27 13-86
paperOut ..... Serial 36 13-95
parallelBits ..... Resources 32 13-93
parallelName ..... Parallel 12 13-86
parallelPort ..... Resources 31 13-93
parallelPrinter ..... Intuition 10 13-67
parent ..... Dos 48 13-15
ParentDir ..... Dos 16 13-18
ParErr ..... Parallel 21 13-86
ParFlags ..... Parallel 25 13-86
ParFlagSet ..... Parallel 26 13-86
parityErr ..... Serial 43 13-95
parityEven ..... Intuition 17 13-67
parityNone ..... Intuition 17 13-67
parityOdd ..... Intuition 19 13-67
partyOdd ..... Serial 31 13-95
partyOn ..... Serial 31 13-95
pb1 ..... Resources 51 13-93
pb16 ..... Resources 53 13-93
pb2 ..... Resources 51 13-93
pb3 ..... Resources 51 13-93
pb4 ..... Resources 51 13-93
pb5 ..... Resources 51 13-93
pb6 ..... Resources 51 13-93
pb7 ..... Resources 51 13-93
pBusy ..... Parallel 27 13-86
pe0 ..... Parallel 22 13-86
pea ..... Assembler 12-14
PenPair ..... Intuition 47 13-69
perf ..... Printer 29 13-88
perf0 ..... Printer 30 13-88
Permit ..... Exec 53 13-26
perVol ..... Audio 21 13- 3
pervol ..... Audio 26 13- 3
pf0 ..... Parallel 25 13-86
pf10 ..... Dos 49 13-12
pf11 ..... Dos 49 13-12
pf12 ..... Dos 49 13-12
pf13 ..... Dos 49 13-12
pf14 ..... Dos 49 13-12
pf15 ..... Dos 49 13-12
pf16 ..... Dos 49 13-12
pf17 ..... Dos 49 13-12
pf18 ..... Dos 49 13-12
pf19 ..... Dos 49 13-12
pf2 ..... Parallel 25 13-86
pf20 ..... Dos 49 13-12
pf21 ..... Dos 50 13-12
pf22 ..... Dos 50 13-12
pf23 ..... Dos 50 13-12
pf24 ..... Dos 50 13-12
pf25 ..... Dos 50 13-12
pf26 ..... Dos 50 13-12
pf27 ..... Dos 50 13-12
pf28 ..... Dos 50 13-12
pf29 ..... Dos 50 13-12
pf2pri ..... Graphics 40 13-38
pf30 ..... Dos 50 13-12
pf31 ..... Dos 51 13-12

pf4	Intuition	4	13-62
pf4	Parallel	25	13-86
pf5	Intuition	4	13-62
pf6	Intuition	4	13-62
pf7	Intuition	4	13-62
pf8	Dos	49	13-12
pf9	Dos	49	13-12
pfba	Graphics	29	13-44
phonErr	Narrator	23	13-84
pi	MathLib0		12-42
pi	MathLibFFP		12-44
pi	MathLibLong		12-46
pica	Intuition	17	13-68
pitchErr	Narrator	25	13-84
pld	Printer	1	13-88
plnCntMsk	Graphics	45	13-38
plnCntShft	Graphics	46	13-38
plu	Printer	57	13-87
pmbAsm	ConUnit	24	13- 8
pmbAwm	ConUnit	25	13- 8
Point	Intuition	43	13-69
pointerpos	InputEvent	15	13-57
pointerSize	Intuition	5	13-67
pointRel	Intuition	33	13-60
PolyDraw	Graphics	34	13-48
portReset	Parallel	22	13-86
portReset	Serial	43	13-95
ports	Hardware	41	13-50
post	Clipboard	15	13- 5
postReset	TrackDisk	6	13-99
PotgoBits	Resources	50	13-93
PotgoBitSet	Resources	54	13-93
potgoName	Resources	47	13-93
Pow	MathIEEEDoubTrans	17	13-82
Pow	MathTrans	18	13-83
pre000ns	Hardware	24	13-50
pre140ns	Hardware	25	13-50
pre280ns	Hardware	26	13-50
pre560ns	Hardware	27	13-50
preComp0	Hardware	18	13-50
preComp1	Hardware	18	13-50
preDrawn	Intuition	33	13-60
Preferences	Intuition	21	13-67
PreferencesPtr	Intuition	33	13-59
prel	Assembler		12-12
primary	Console	21	13- 6
primaryClip	Clipboard	42	13- 5
PrinterClass	PrtBase	5	13-91
PrinterClassSet	PrtBase	6	13-91
PrinterData	PrtBase	37	13-90
PrinterDataPtr	PrtBase	4	13-91
PrinterExtendedData	PrtBase	24	13-91
PrinterExtendedDataPtr	PrtBase	45	13-91
printerName	Printer	16	13-87
PrinterPort	Intuition	10	13-67
PrinterSegment	PrtBase	46	13-91
PrinterSegmentPtr	PrtBase	36	13-90
PrinterType	Intuition	11	13-67
PrintIText	Intuition	43	13-74
PROCESS	Coroutines		12-20
Process	Dos	26	13-14
process	Exec	17	13-19
ProcessId	Dos	24	13-14
ProcessPtr	Dos	25	13-14
procTime	Exec	6	13-21
Procure	Exec	54	13-26
project	Workbench	23	13-101
prop0	Printer	17	13-88
prop1	Printer	16	13-88
prop2	Printer	15	13-88
propBorderless	Intuition	4	13-62
propGadget	Intuition	16	13-61
PropInfo	Intuition	16	13-62
PropInfoFlags	Intuition	3	13-62

PropInfoFlagSet ..... Intuition 7 13-62
PropInfoPtr ..... Intuition 34 13-59
proportional ..... Graphics 46 13-42
ProtectionFlags ..... Dos 47 13-12
ProtectionFlagSet ..... Dos 52 13-12
protStatus ..... TrackDisk 21 13-98
prtCommand ..... Printer 18 13-87
prtrBusy ..... Hardware 56 13-53
prtrPOut ..... Hardware 56 13-53
prtrSel ..... Hardware 56 13-53
pSel ..... Parallel 27 13-86
pstd ..... Workbench 25 13-101
public ..... Exec 6 13-20
pure ..... Dos 48 13-12
PutDiskObject ..... Icon 37 13-55
PutIcon ..... Icon 32 13-55
PutMsg ..... Exec 57 13-26
PutSeed ..... RandomNumber 12-47
PutWBObject ..... Icon 26 13-55
QBlit ..... Graphics 37 13-48
QBSBlit ..... Graphics 38 13-48
Qualifiers ..... InputEvent 46 13-57
QualifierSet ..... InputEvent 51 13-57
query ..... Parallel 13 13-86
query ..... Serial 15 13-95
queued ..... Parallel 27 13-86
queued ..... Serial 22 13-95
queuedBrk ..... Serial 31 13-95
QueuePacket ..... Dos 19 13-18
quick ..... Exec 45 13-22
QumeLP20 ..... Intuition 13 13-67
quoted ..... Arguments 12- 3
radBoogie ..... Parallel 25 13-86
radBoogie ..... Serial 31 13-95
rAlt ..... InputEvent 47 13-57
Random ..... RandomNumber 12-47
RasInfo ..... Graphics 56 13-44
RasInfoPtr ..... Graphics 33 13-36
RasSize ..... GfxMacros 15 13-35
raster ..... Hardware 32 13-50
RastPort ..... Graphics 55 13-42
RastPortFlags ..... Graphics 50 13-42
RastPortFlagSet ..... Graphics 54 13-42
RastPortPtr ..... Graphics 34 13-36
rateErr ..... Narrator 24 13-84
RawDoFmt ..... Exec 2 13-27
RawIOInit ..... Exec 7 13-27
rawkey ..... InputEvent 15 13-57
rawKey ..... Intuition 34 13-63
RawKeyConvert ..... Console 16 13- 7
RawMayGetChar ..... Exec 8 13-27
rawmouse ..... InputEvent 15 13-57
RawPutChar ..... Exec 9 13-27
rawRead ..... TrackDisk 22 13-98
rawWrite ..... Printer 17 13-87
rawWrite ..... TrackDisk 23 13-98
rbf ..... Hardware 42 13-50
rButton ..... InputEvent 38 13-57
rCommand ..... InputEvent 47 13-57
Read ..... Dos 21 13-18
read ..... Dos 33 13-15
read ..... Exec 29 13-22
read ..... FileSystem 12-28
Read ..... InOut 12-36
Read ..... Terminal 12-59
readBits ..... Intuition 38 13-68
readBreak ..... Serial 37 13-95
ReadByteBlock ..... FileSystem 12-29
ReadBytes ..... FileSystem 12-29
ReadBytes ..... InOut 12-36
ReadCard ..... InOut 12-36
ReadChar ..... FileSystem 12-29
readErr ..... FileSystem 12-28
readEvent ..... GamePort 13 13-34

readEvent ..... Keyboard 13 13-77
ReadExpansionByte ..... Expansion 43 13-32
ReadExpansionRom ..... Expansion 45 13-32
ReadFileName ..... FileNames 12-26
ReadInt ..... InOut 12-36
ReadLn ..... Terminal 12-59
ReadLongCard ..... InOut 12-36
ReadLongInt ..... InOut 12-36
readMatrix ..... Keyboard 14 13-77
readOnly ..... Dos 19 13-12
ReadPixel ..... Graphics 39 13-48
ReadProc ..... FileNames 12-26
ReadProc ..... Scan 12-51
readProt ..... Dos 48 13-12
readProtected ..... Dos 4 13-14
ReadReal ..... FFPInOut 12-24
ReadReal ..... LongRealInOut 12-41
ReadReal ..... RealInOut 12-50
ReadString ..... InOut 12-36
readWrite ..... Dos 18 13-12
ready ..... Exec 9 13-21
readyToSend ..... Serial 37 13-95
RealToStr ..... FFPConversions 12-22
RealToStr ..... LongRealConversions 12-39
RealToStr ..... RealConversions 12-48
recoveryAlert ..... Intuition 57 13-68
Rectangle ..... Graphics 47 13-36
RectanglePtr ..... Graphics 35 13-36
RectFill ..... Graphics 42 13-48
red ..... Console 27 13- 6
redBg ..... Console 36 13- 6
RefreshGadgets ..... Intuition 47 13-74
RefreshGList ..... Intuition 51 13-74
refreshwindow ..... InputEvent 17 13-57
refreshWindow ..... Intuition 33 13-63
RefreshWindowFrame ..... Intuition 55 13-74
Region ..... Graphics 45 13-43
RegionPtr ..... Graphics 36 13-36
RegionRectangle ..... Graphics 40 13-43
RegionRectanglePtr ..... Graphics 37 13-36
relativeMouse ..... InputEvent 50 13-57
ReleaseConfigBinding ..... Expansion 48 13-32
ReleaseSemaphore ..... Exec 10 13-27
ReleaseSemaphoreList ..... Exec 12 13-27
relJoystick ..... GamePort 30 13-34
relVerify ..... Intuition 5 13-61
RemakeDisplay ..... Intuition 56 13-74
RemBob ..... GfxMacros 14 13-35
remChangeInt ..... TrackDisk 27 13-98
RemConfigDev ..... Expansion 49 13-32
RemDevice ..... Exec 13 13-27
Remember ..... Intuition 50 13-68
RememberPtr ..... Intuition 35 13-59
RemFont ..... Graphics 47 13-48
remHandler ..... Input 14 13-56
RemHead ..... Exec 14 13-27
RemIBob ..... Graphics 48 13-48
RemICRVector ..... Resources 29 13-92
RemIntServer ..... Exec 15 13-27
RemLibrary ..... Exec 17 13-27
Remove ..... Exec 19 13-27
remove ..... TrackDisk 18 13-98
removed ..... Exec 9 13-21
removed ..... Graphics 48 13-42
RemoveDebugProc ..... Arts 12- 9
RemoveGadget ..... Intuition 57 13-74
RemoveGList ..... Intuition 3 13-75
RemoveTermProc ..... Arts 12- 9
RemPort ..... Exec 20 13-27
remResetHandler ..... Keyboard 16 13-77
RemResource ..... Exec 21 13-27
RemSemaphore ..... Exec 22 13-27
RemTail ..... Exec 24 13-27
RemTask ..... Exec 25 13-27

RemVSprite ..... Graphics 51 13-48
Rename ..... Dos 24 13-18
renameAcrossDevices ..... Dos 52 13-13
renameDisk ..... Dos 31 13-15
renameObject ..... Dos 36 13-15
repeat ..... InputEvent 48 13-57
replMd ..... Windows 12-64
replyMsg ..... Exec 17 13-19
ReplyMsg ..... Exec 26 13-27
ReportMouse ..... Intuition 6 13-75
reportMouse ..... Intuition 35 13-64
reqActive ..... Intuition 34 13-60
reqClear ..... InputEvent 42 13-57
reqClear ..... Intuition 35 13-63
reqGadget ..... Intuition 26 13-61
reqOffWindow ..... Intuition 34 13-60
reqSet ..... InputEvent 43 13-57
reqSet ..... Intuition 34 13-63
Request ..... Intuition 8 13-75
Requester ..... Arts 12- 9
requester ..... InputEvent 16 13-57
Requester ..... Intuition 37 13-60
RequesterFlags ..... Intuition 32 13-60
RequesterFlagSet ..... Intuition 36 13-60
RequesterPtr ..... Intuition 36 13-59
reqVerify ..... Intuition 35 13-63
Res ..... Intuition 10 13-69
Reschedule ..... Exec 27 13-27
resCount ..... Intuition 13 13-69
reserved ..... Exec 43 13-21
reserved8 ..... Exec 52 13-23
reserved9 ..... Exec 53 13-23
reset ..... Assembler 12-14
reset ..... Exec 28 13-22
reset ..... Expansion 12 13-31
resetHandlerDone ..... Keyboard 17 13-77
Resident ..... Exec 33 13-23
ResidentFlags ..... Exec 30 13-23
ResidentFlagSet ..... Exec 31 13-23
ResidentPtr ..... Exec 32 13-23
resource ..... Exec 17 13-19
ResourceTypes ..... Resources 30 13-93
Response ..... FileSystem 12-28
ResponseText ..... FileMessage 12-25
RethinkDisplay ..... Intuition 10 13-75
returnVal ..... Arts 12- 8
revPath ..... Graphics 46 13-42
rf1 ..... Exec 30 13-23
rf10 ..... Intuition 34 13-60
rf11 ..... Intuition 34 13-60
rf2 ..... Exec 30 13-23
rf3 ..... Exec 30 13-23
rf3 ..... Intuition 33 13-60
rf4 ..... Exec 30 13-23
rf4 ..... Intuition 33 13-60
rf5 ..... Exec 30 13-23
rf5 ..... Intuition 33 13-60
rf6 ..... Exec 30 13-23
rf6 ..... Intuition 33 13-60
rf7 ..... Intuition 33 13-60
rf8 ..... Intuition 33 13-60
rf9 ..... Intuition 33 13-60
ri ..... Printer 25 13-87
rightBorder ..... Intuition 5 13-61
rightButton ..... InputEvent 49 13-57
rightHit ..... Graphics 45 13-37
rin ..... Printer 22 13-87
ringtrigger ..... Graphics 57 13-38
ris ..... Printer 21 13-87
rmbTrap ..... Intuition 37 13-64
rms ..... Printer 33 13-88
RND ..... RandomNumber 12-47
robotic ..... Narrator 35 13-84
rolB ..... Assembler 12-14

roliB ..... Assembler 12-14
roliL ..... Assembler 12-14
roliW ..... Assembler 12-14
rolL ..... Assembler 12-14
rolmW ..... Assembler 12-14
rolW ..... Assembler 12-14
romFont ..... Graphics 46 13-42
RootNode ..... Dos 18 13-16
RootNodePtr ..... Dos 57 13-15
rorB ..... Assembler 12-14
roriB ..... Assembler 12-14
roriL ..... Assembler 12-14
roriW ..... Assembler 12-14
rorL ..... Assembler 12-14
rormW ..... Assembler 12-14
rorW ..... Assembler 12-14
roxlB ..... Assembler 12-14
roxliB ..... Assembler 12-14
roxliL ..... Assembler 12-14
roxliW ..... Assembler 12-14
roxlL ..... Assembler 12-14
roxlmW ..... Assembler 12-14
roxlW ..... Assembler 12-14
roxrB ..... Assembler 12-14
roxriB ..... Assembler 12-14
roxriL ..... Assembler 12-14
roxriW ..... Assembler 12-14
roxrL ..... Assembler 12-14
roxrmW ..... Assembler 12-15
roxrW ..... Assembler 12-14
rpf4 ..... Graphics 51 13-42
rpf6 ..... Graphics 52 13-42
rpf7 ..... Graphics 52 13-42
rpf8 ..... Graphics 53 13-42
rpLock ..... Intuition 28 13-69
rs ..... ASCII 12-11
rsc ..... Arts 12- 7
rShift ..... InputEvent 47 13-57
rtd ..... Assembler 12-15
rte ..... Assembler 12-15
rtr ..... Assembler 12-15
rts ..... Assembler 12-15
run ..... Exec 9 13-21
rwDir ..... Parallel 27 13-86
SatisfyMsg ..... Clipboard 45 13- 5
SatisfyMsgPtr ..... Clipboard 50 13- 5
saveBack ..... Graphics 9 13-39
saveBob ..... Graphics 36 13-39
savePreserve ..... Graphics 37 13-39
sbc ..... Printer 35 13-87
sbcdB ..... Assembler 12-15
sbcdmB ..... Assembler 12-15
ScanString ..... Scan 12-51
scc ..... Assembler 12-15
Schedule ..... Exec 28 13-27
Screen ..... Intuition 25 13-66
screenBehind ..... Intuition 44 13-64
ScreenFlags ..... Intuition 42 13-64
ScreenFlagSet ..... Intuition 45 13-64
screenMode ..... Dos 54 13-15
ScreenPtr ..... Intuition 37 13-59
screenQuiet ..... Intuition 44 13-64
ScreenToBack ..... Intuition 11 13-75
ScreenToFront ..... Intuition 12 13-75
scrGadget ..... Intuition 28 13-61
script ..... Dos 48 13-12
Scroll ..... Windows 12-64
scrolling ..... Windows 12-64
ScrollLayer ..... Layers 51 13-80
ScrollRaster ..... Graphics 52 13-48
ScrollVPort ..... Graphics 2 13-49
scs ..... Assembler 12-15
sDownBack ..... Intuition 24 13-61
sDownBackGadget ..... Intuition 18 13-69

sDragGadget	Intuition	19	13-69
sDragging	Intuition	20	13-61
secShift	TrackDisk	42	13-98
sector	TrackDisk	41	13-98
Seek	Dos	5	13-18
seek	TrackDisk	16	13-98
seekErr	FileSystem		12-28
seekError	Dos	56	13-13
seekError	TrackDisk	1	13-99
select	Serial	36	13-95
selectDown	Intuition	44	13-63
selected	Intuition	1	13-61
selectUp	Intuition	43	13-63
Semaphore	Exec	13	13-23
semaphore	Exec	18	13-19
SemaphoreRequest	Exec	17	13-23
SendIO	Exec	29	13-27
seq	Assembler		12-15
SerFlags	Serial	30	13-95
SerFlagSet	Serial	34	13-95
serialBits	Resources	31	13-93
serialName	Serial	14	13-95
serialPort	Resources	31	13-93
serialPrinter	Intuition	10	13-67
SerParShk	Intuition	16	13-67
SerParShkSet	Intuition	20	13-67
SetAfPen	GfxMacros	25	13-35
SetAPen	Graphics	3	13-49
SetBPen	Graphics	5	13-49
SetClip	Windows		12-64
setclr	Hardware	33	13-53
SetCollision	Graphics	7	13-49
SetColor	Windows		12-64
SetComment	Dos	8	13-18
setComment	Dos	47	13-15
setCType	GamePort	15	13-34
SetCurrentBinding	Expansion	50	13-32
setDate	Dos	53	13-15
setDefaultKeyMap	Console	20	13- 6
SetDMRequest	Intuition	13	13-75
SetDrMd	Graphics	10	13-49
SetDrPt	GfxMacros	23	13-35
SetExcept	Exec	30	13-27
SetFont	Graphics	12	13-49
SetFunction	Exec	32	13-27
SetICR	Resources	32	13-92
SetIntVector	Exec	36	13-27
setKeyMap	Console	18	13- 6
setMap	Dos	26	13-15
SetMenuStrip	Intuition	16	13-75
SetMode	Windows		12-64
setMPort	Input	18	13-56
setMTrig	Input	20	13-56
setMType	Input	19	13-56
SetOPen	GfxMacros	22	13-35
setParams	Parallel	14	13-86
setParams	Serial	17	13-95
setPeriod	Input	17	13-56
SetPointer	Intuition	18	13-75
SetPos	FileSystem		12-29
SetPos	Windows		12-64
setPrec	Audio	19	13- 3
SetPrefs	Intuition	24	13-75
setProtect	Dos	40	13-15
SetProtection	Dos	10	13-18
SetRast	Graphics	14	13-49
SetRGB4	Graphics	16	13-49
SetRGB4CM	Graphics	21	13-49
SetScreen	Windows		12-64
SetSignal	Exec	39	13-27
SetSoftStyle	Graphics	26	13-49
SetSR	Exec	41	13-27
setSysTime	Timer	17	13-97
SetTaskPri	Exec	43	13-27

setThresh	Input	16	13-56
setTrigger	GamePort	17	13-34
SetWindowTitles	Intuition	27	13-75
SetWrMsk	GfxMacros	24	13-35
sevenWire	Serial	31	13-95
sexErr	Narrator	26	13-84
sf	Assembler		12-15
sf1	Intuition	43	13-64
sf2	Intuition	43	13-64
sf3	Intuition	43	13-64
sfc	Printer	34	13-87
sge	Assembler		12-15
sgr0	Printer	27	13-87
sgr1	Printer	32	13-87
sgr22	Printer	33	13-87
sgr23	Printer	29	13-87
sgr24	Printer	31	13-87
sgr3	Printer	28	13-87
sgr4	Printer	30	13-87
sgt	Assembler		12-15
shadeBW	Intuition	28	13-68
shadeColor	Intuition	30	13-68
shadeGreyscale	Intuition	29	13-68
shakeNone	Intuition	17	13-67
shakeRts	Intuition	17	13-67
shakeXon	Intuition	17	13-67
shared	Parallel	25	13-86
shared	Serial	31	13-95
sharedLock	Dos	30	13-12
shi	Assembler		12-15
shift	KeyMap	14	13-78
shorp0	Printer	37	13-87
shorp1	Printer	39	13-87
shorp2	Printer	38	13-87
shorp3	Printer	41	13-87
shorp4	Printer	40	13-87
shorp5	Printer	43	13-87
shorp6	Printer	42	13-87
ShortSet	Hardware	9	13-54
ShowTitle	Intuition	31	13-75
showTitle	Intuition	43	13-64
sht	Graphics	51	13-37
shutup	Expansion	57	13-31
si	ASCII		12-11
sigAbort	Exec	37	13-21
sigBlit	Exec	39	13-21
sigChild	Exec	38	13-21
sigDos	Exec	40	13-21
signal	Exec	38	13-20
Signal	Exec	45	13-27
signalSem	Exec	19	13-19
SignalSemaphore	Exec	21	13-23
SignalSemaphorePtr	Exec	29	13-23
signFlag	Hardware	11	13-51
simpleRefresh	Intuition	35	13-64
SimpleSprite	Graphics	49	13-43
SimpleSpritePtr	Graphics	38	13-36
Sin	MathIEEEDoubTrans	18	13-82
sin	MathLib0		12-42
sin	MathLibFFP		12-44
sin	MathLibLong		12-46
Sin	MathTrans	19	13-83
Sincos	MathIEEEDoubTrans	19	13-82
Sincos	MathTrans	20	13-83
single	Intuition	37	13-68
Sinh	MathIEEEDoubTrans	21	13-82
Sinh	MathTrans	21	13-83
sixLPI	Intuition	22	13-68
sizeBBottom	Intuition	35	13-64
sizeBRight	Intuition	34	13-64
sizeGadget	Intuition	17	13-69
SizeLayer	Layers	53	13-80
sizeVerify	Intuition	33	13-63
sizewindow	InputEvent	16	13-57

SizeWindow	Intuition	33	13-75
sizing	Intuition	18	13-61
sizing	Windows		12-64
sle	Assembler		12-15
slotMask	Expansion	47	13-30
slotShift	Expansion	48	13-30
slotSize	Expansion	46	13-30
slpp	Printer	28	13-88
slrm	Printer	37	13-88
sls	Assembler		12-15
slt	Assembler		12-15
smi	Assembler		12-15
sne	Assembler		12-15
so	ASCII		12-11
softInt	Exec	17	13-19
softint	Exec	38	13-20
softint	Hardware	41	13-50
SoftIntList	Exec	1	13-20
soh	ASCII		12-11
SortGList	Graphics	30	13-49
sp	ASCII		12-11
sp	Hardware	33	13-53
Special	Printer	12	13-89
SpecialSet	Printer	16	13-89
spl	Assembler		12-15
sprite	Hardware	31	13-50
spriteAttached	Graphics	36	13-43
sprites	Graphics	30	13-44
sps3	Intuition	17	13-67
Sqrt	MathIEEEDoubTrans	22	13-82
sqrt	MathLib0		12-42
sqrt	MathLibFFP		12-44
sqrt	MathLibLong		12-46
Sqrt	MathTrans	22	13-83
srcA	Hardware	56	13-50
srcB	Hardware	56	13-50
srcC	Hardware	56	13-50
st	Assembler		12-15
stackChk	Exec	6	13-21
stackSize	Arts		12- 8
stackTop	Arts		12- 8
StandardPacket	Dos	17	13-15
start	Exec	34	13-22
start	Resources	51	13-93
Startup	Arts		12- 8
startupMsg	Arts		12- 8
Status	Parallel	27	13-86
Status	Serial	35	13-95
StatusSet	Parallel	28	13-86
StatusSet	Serial	40	13-95
stbm	Printer	36	13-88
stdScreenHeight	Intuition	50	13-64
StkChk	Arts		12- 9
stkOvl	Arts		12- 7
stkSize	PrtBase	33	13-90
stop	Assembler		12-15
stop	Exec	33	13-22
stopBits	Intuition	40	13-68
strGadget	Intuition	17	13-61
string	KeyMap	14	13-78
stringCenter	Intuition	6	13-61
StringInfo	Intuition	30	13-62
StringInfoPtr	Intuition	38	13-59
stringRight	Intuition	7	13-61
StrPtr	FileMessage		12-25
StrToReal	FFPConversions		12-22
StrToReal	LongRealConversions		12-39
StrToReal	RealConversions		12-48
StrToVal	Conversions		12-17
stx	ASCII		12-11
sub	ASCII		12-11
subaL	Assembler		12-15
subaW	Assembler		12-15
subB	Assembler		12-15

subiB	Assembler		12-15
subiL	Assembler		12-15
subiW	Assembler		12-15
subL	Assembler		12-15
submB	Assembler		12-15
submL	Assembler		12-15
submW	Assembler		12-15
subqB	Assembler		12-15
subqL	Assembler		12-15
subqW	Assembler		12-15
SubTime	Timer	38	13-97
subW	Assembler		12-15
subxB	Assembler		12-15
subxL	Assembler		12-15
subxmB	Assembler		12-15
subxmL	Assembler		12-15
subxmW	Assembler		12-15
subxW	Assembler		12-15
sud	Hardware	20	13-51
sul	Hardware	19	13-51
SumKickData	Exec	47	13-27
SumLibrary	Exec	48	13-27
summing	Exec	53	13-21
sumUsed	Exec	53	13-21
superBitMap	Intuition	35	13-64
SuperState	Exec	49	13-27
superUnused	Intuition	49	13-64
Supervisor	Exec	50	13-27
sUpFront	Intuition	22	13-61
sUpFrontGadget	Intuition	18	13-69
sus0	Printer	56	13-87
sus1	Printer	53	13-87
sus2	Printer	52	13-87
sus3	Printer	55	13-87
sus4	Printer	54	13-87
svc	Assembler		12-15
svs	Assembler		12-15
SwapBitsRastPortClipRect	Layers	55	13-80
swapW	Assembler		12-15
switch	Exec	6	13-21
Switch	Exec	51	13-27
syn	ASCII		12-11
syncCycle	Audio	27	13- 3
SyncSBitMap	Graphics	31	13-49
SysErr	Arts		12- 7
sysGadget	Intuition	29	13-61
sysRequest	Intuition	34	13-60
system	Arts		12- 7
SystemError	Arts		12- 9
ta	Hardware	33	13-53
tallDot	Graphics	46	13-42
Tan	MathIEEEDoubTrans	23	13-82
Tan	MathTrans	23	13-83
Tanh	MathIEEEDoubTrans	24	13-82
Tanh	MathTrans	24	13-83
tasB	Assembler		12-15
Task	Exec	10	13-21
task	Exec	16	13-19
TaskArray	Dos	11	13-16
TaskArrayPtr	Dos	17	13-16
TaskFlags	Exec	5	13-21
TaskFlagSet	Exec	8	13-21
TaskPtr	Exec	39	13-20
TaskState	Exec	9	13-21
taskTableFull	Dos	35	13-13
tb	Hardware	33	13-53
tbc0	Printer	42	13-88
tbc1	Printer	44	13-88
tbc3	Printer	43	13-88
tbc4	Printer	45	13-88
tbcall	Printer	46	13-88
tbcHClrTab	Console	7	13- 7
tbcHClrTabsAll	Console	8	13- 7
tbe	Hardware	41	13-50

tbsall	Printer	47	13-88
TDUPublicUnit	TrackDisk	15	13-99
termCh	InOut		12-36
Terminate	Arts		12- 9
TermProcedure	Arts		12- 9
Text	Graphics	32	13-49
TextAttr	Graphics	56	13-43
TextAttrPtr	Graphics	39	13-36
TextFont	Graphics	5	13-44
TextFontPtr	Graphics	40	13-36
TextLength	Graphics	35	13-49
tf1	Exec	6	13-21
tf2	Exec	6	13-21
tf3	Exec	6	13-21
ThinLayerInfo	Layers	1	13-81
ticksPerSecond	Dos	35	13-12
Tieee	MathIEEEDoubTrans	25	13-82
Tieee	MathTrans	25	13-83
timer	Dos	49	13-15
timer	InputEvent	15	13-57
timer	Workbench	25	13-101
timerErr	Serial	43	13-95
timerName	Timer	14	13-97
TimeVal	Timer	22	13-97
TimeValPtr	Timer	26	13-97
TmpRas	Graphics	22	13-42
TmpRasPtr	Graphics	41	13-36
tms	Printer	34	13-88
toggleSelect	Intuition	6	13-61
tooFewSecs	TrackDisk	54	13-98
tool	Workbench	23	13-101
toolExit	Workbench	25	13-101
tooManyLevels	Dos	54	13-13
topazEighty	Intuition	6	13-67
topazSixty	Intuition	7	13-67
topBorder	Intuition	6	13-61
topHit	Graphics	42	13-37
trackDiskName	TrackDisk	14	13-98
TRANSFER	Coroutines		12-20
Translate	Translator	15	13-100
trap	Arts		12- 7
trap	Assembler		12-15
trapv	Assembler		12-15
tss	Printer	18	13-88
tstB	Assembler		12-15
tstL	Assembler		12-15
tstW	Assembler		12-15
typeBit	Expansion	56	13-30
TypeOfMem	Exec	52	13-27
Types	KeyMap	36	13-78
typeSize	Expansion	57	13-30
TypesPtr	KeyMap	37	13-78
typrMask	Expansion	55	13-30
uartBrk	Hardware	18	13-50
UByte	Exec	14	13-19
UByte	Hardware	13	13-50
UCopList	Graphics	27	13-38
UCopListPtr	Graphics	42	13-36
UCopperListInit	Graphics	38	13-49
underlined	Graphics	43	13-42
underscore	Console	24	13- 6
unimpl	Narrator	20	13-84
Unit	Exec	18	13-22
unitErr	Narrator	18	13-84
UnitFlags	Exec	15	13-22
UnitFlagSet	Exec	16	13-22
UnitPtr	Exec	17	13-22
unknown	Exec	16	13-19
unlk	Assembler		12-15
UnLoadSeg	Dos	26	13-18
UnLock	Dos	27	13-18
UnlockIBase	Intuition	36	13-75
UnlockLayer	Layers	3	13-81
UnlockLayerInfo	Layers	4	13-81

UnlockLayerRom ..... Graphics 41 13-49
UnlockLayers ..... Layers 5 13-81
unreadableDisk ..... Dos 29 13-13
update ..... Exec 31 13-22
upFrontGadget ..... Intuition 17 13-69
UpfrontLayer ..... Layers 6 13-81
upKeys ..... GamePort 22 13-34
upPrefix ..... InputEvent 23 13-57
us ..... ASCII 12-11
use0p1 ..... Hardware 17 13-50
use0v1 ..... Hardware 17 13-50
use1p2 ..... Hardware 17 13-50
use1v2 ..... Hardware 17 13-50
use2p3 ..... Hardware 17 13-50
use2v3 ..... Hardware 17 13-50
use3pn ..... Hardware 17 13-50
use3vn ..... Hardware 17 13-50
userDef ..... Exec 45 13-21
UserState ..... Exec 53 13-27
usLegal ..... Intuition 32 13-68
usLetter ..... Intuition 31 13-68
Vacate ..... Exec 54 13-27
validated ..... Dos 26 13-13
validating ..... Dos 25 13-13
ValToStr ..... Conversions 12-17
vanilla ..... KeyMap 24 13-78
vanillaKey ..... Intuition 37 13-63
VBeamPos ..... Graphics 42 13-49
vbit ..... Assembler 12-12
vBlank ..... Timer 19 13-97
vectSize ..... Exec 42 13-21
verp0 ..... Printer 26 13-88
verp1 ..... Printer 27 13-88
vertb ..... Hardware 41 13-50
vf4 ..... Graphics 9 13-39
vf5 ..... Graphics 9 13-39
vf6 ..... Graphics 9 13-39
vf7 ..... Graphics 9 13-39
View ..... Graphics 48 13-44
ViewAddress ..... Intuition 37 13-75
viewLock ..... Intuition 26 13-69
ViewModes ..... Graphics 28 13-44
ViewModeSet ..... Graphics 32 13-44
ViewPort ..... Graphics 33 13-44
ViewPortAddress ..... Intuition 38 13-75
ViewPortPtr ..... Graphics 44 13-36
ViewPtr ..... Graphics 43 13-36
vm0 ..... Graphics 29 13-44
vm12 ..... Graphics 30 13-44
vm3 ..... Graphics 29 13-44
vm4 ..... Graphics 29 13-44
vm5 ..... Graphics 29 13-44
vm9 ..... Graphics 30 13-44
volErr ..... Narrator 29 13-84
volume ..... Dos 51 13-16
vpHide ..... Graphics 30 13-44
vposrlof ..... Graphics 55 13-38
vrtclPos ..... Graphics 51 13-38
vrtclPosShift ..... Graphics 52 13-38
vSizeBits ..... Hardware 48 13-50
vSizeMask ..... Hardware 50 13-50
vsOverflow ..... Graphics 10 13-39
vsprite ..... Graphics 9 13-39
VSprite ..... Graphics 12 13-39
VSpriteFlags ..... Graphics 9 13-39
VSpriteFlagSet ..... Graphics 11 13-39
VSpritePtr ..... Graphics 45 13-36
vt ..... ASCII 12-11
vts ..... Printer 41 13-88
wait ..... Exec 9 13-21
Wait ..... Exec 55 13-27
wait ..... Graphics 49 13-37
WaitBlit ..... Graphics 43 13-49
WaitBOVP ..... Graphics 44 13-49

waitChar	Dos	39	13-15
waitCloseGadget	Terminal		12-59
waitCycle	Audio	23	13- 3
WaitForChar	Dos	28	13-18
WaitIO	Exec	56	13-27
WaitPort	Exec	57	13-27
WaitTOF	Graphics	45	13-49
warn	Dos	13	13-14
wb	PrtBase	18	13-91
wb0	Workbench	23	13-101
WBArg	Workbench	31	13-101
WBArgPtr	Workbench	29	13-101
WBArgumentsPtr	Workbench	35	13-101
wbenchClose	Intuition	9	13-64
wbenchMessage	Intuition	36	13-63
wbenchOpen	Intuition	8	13-64
wbenchScreen	Intuition	43	13-64
WBenchToBack	Intuition	40	13-75
WBenchToFront	Intuition	41	13-75
wbenchWindow	Intuition	38	13-64
WBObjectType	Workbench	22	13-101
wbStarted	Arts		12- 8
WBStartup	Workbench	36	13-101
WBStartupPtr	Workbench	30	13-101
wDownBack	Intuition	23	13-61
wDragging	Intuition	19	13-61
wf18	Intuition	37	13-64
wf19	Intuition	37	13-64
wf20	Intuition	37	13-64
wf21	Intuition	37	13-64
wf22	Intuition	37	13-64
wf23	Intuition	38	13-64
wf27	Intuition	38	13-64
wf28	Intuition	38	13-64
wf29	Intuition	39	13-64
wf30	Intuition	39	13-64
wf31	Intuition	40	13-64
WhichLayer	Layers	7	13-81
white	Console	33	13- 6
whiteBg	Console	42	13- 6
wideDot	Graphics	46	13-42
Window	Intuition	32	13-65
Window	Windows		12-64
windowActive	Intuition	36	13-64
windowClose	Intuition	34	13-64
windowDepth	Intuition	34	13-64
windowDrag	Intuition	34	13-64
WindowFlags	Intuition	33	13-64
WindowFlagSet	Intuition	41	13-64
WindowLimits	Intuition	50	13-75
WindowPtr	Intuition	39	13-59
windowRefresh	Intuition	38	13-64
windowSizing	Intuition	34	13-64
windowTicked	Intuition	38	13-64
WindowToBack	Intuition	56	13-75
WindowToFront	Intuition	57	13-75
WinGad	Windows		12-64
WinGadSet	Windows		12-64
wordSync	Hardware	18	13-50
wordWide	Expansion	34	13-31
Write	Dos	30	13-18
write	Dos	32	13-15
write	Exec	30	13-22
write	FileSystem		12-28
Write	InOut		12-36
Write	Terminal		12-59
writeBits	Intuition	39	13-68
WriteByteBlock	FileSystem		12-29
WriteBytes	FileSystem		12-29
WriteBytes	InOut		12-36
WriteC	Windows		12-64
WriteCard	InOut		12-36
WriteChar	FileSystem		12-29
writeErr	FileSystem		12-28

writeEvent ..... Input 15 13-56
WriteExpansionByte ..... Expansion 53 13-32
WriteHex ..... InOut 12-36
WriteInt ..... InOut 12-36
WriteL ..... Windows 12-64
WriteLn ..... InOut 12-36
WriteLn ..... Terminal 12-59
writeMessage ..... Audio 29 13- 3
WriteOct ..... InOut 12-36
WritePixel ..... Graphics 46 13-49
WritePotgo ..... Resources 4 13-94
WriteProc ..... FileNames 12-26
writeProt ..... Dos 48 13-12
writeProt ..... TrackDisk 56 13-98
writeProtect ..... Dos 24 13-13
writeProtected ..... Dos 3 13-14
writeProtected ..... FileSystem 12-28
WriteReal ..... FFPInOut 12-24
WriteReal ..... LongRealInOut 12-41
WriteReal ..... RealInOut 12-50
WriteS ..... Windows 12-64
WriteString ..... InOut 12-36
WriteString ..... Terminal 12-59
wroteBreak ..... Serial 37 13-95
wTractor ..... Intuition 34 13-68
wUpFront ..... Intuition 21 13-61
xDisabled ..... Serial 33 13-95
xOffRead ..... Serial 39 13-95
xOffWrite ..... Serial 38 13-95
xorMd ..... Windows 12-64
XorRectRegion ..... Graphics 49 13-49
XorRegionRegion ..... Graphics 52 13-49
XYtoPos ..... Windows 12-64
yellow ..... Console 29 13- 6
yellowBg ..... Console 38 13- 6
ymc ..... PrtBase 13 13-91
ymcb ..... PrtBase 15 13-91
ymcBw ..... PrtBase 14 13-91
zbit ..... Assembler 12-12

M2 AMIGA enthält eine vollständige Software-Entwicklungsumgebung für den professionellen Modula-2-Programmierer. Dieses Software-Entwicklungssystem ist das effizienteste Werkzeug für die leistungsfähige Sprache Modula-2.

M2 AMIGA enthält:

- einen Einpass-Compiler, der in kürzester Zeit aus normalen ASCII-Dateien schnellen 68000-Maschinencode erzeugt.
- Compilerunterstützung von Amigaspezifischem wie das Lesen und Schreiben von Registern, Inline 68000'er-Anweisungen, FFP-Darstellung von REAL-Zahlen (nebst IEEE) wie auch Typen doppelter Genauigkeit (LONGINT, LONGCARD, LONGREAL). ROM-Aufrufe (EXEC, Intuition, usw.) benötigen keinen Zwischencode.
- einen Editor, der Fehler im Klartext anzeigen kann.
- einen Linker, der die compilierten Module in Sekunden zu einem lauffähigen Programm zusammenbindet.
- alle Schnittstellen zum Amiga-Betriebssystem.
- zahlreiche Grundbibliotheken: Arguments, ASCII, Conversions, Coroutines, FileNames, FileMessage, FileSystem, FFPConversions, FFPInOut, Heap, InOut, LongRealInOut, MathLib0, MathLibLong, Processes, RandomNumbers, RealConversions, RealInOut, Storage, Strings, Terminal, TextWindows.

A+L AG

ISBN 3-907017-20-X

Dieses Modula-2 Software-Entwicklungssystem wurde selbst mit Modula-2 auf dem Amiga entwickelt.